Smart Card DBMS: where are we now?

Nicolas Anciaux, Luc Bouganim, Philippe Pucheral

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Smart Card DBMS: where are we now?

Nicolas Anciaux, Luc Bouganim, Philippe Pucheral – SMIS Project

N° 00080840
Juin 2006
Thème SYM

ISSN 0249-6399

Smart Card DBMS: where are we now?

Nicolas Anciaux[1], Luc Bouganim[2], Philippe Pucheral[3]

Thème SYM – Systèmes symboliques
Projet SMIS

Rapport de recherche n° 00080840 – Juin 2006 – 37 pages

Abstract: Smart card is today the most widespread secured portable computing device. Four years ago, we addressed the problem of scaling down database techniques for the smart card and we proposed the design of what we called a PicoDBMS, a full-fledged database system embedded in a smart card. Since then, thanks to the hardware progress and to the joint implementation efforts of our team and our industrial partner, this utopian design gave birth to a complete prototype running on an experimental smart card platform. This paper revisits the problem statement in the light of the hardware and applications evolution. Then, it introduces a benchmark dedicated to Pico–style databases and provides an extensive performance analysis of our prototype, discussing lessons learned at experimentation time and helping selecting the appropriate storage and indexation model for a given class of embedded applications. Finally, it draws new research perspectives for data management on secured chips (smart cards, USB dongles, multimedia rendering devices, smart objects in an ambient intelligence surrounding).

Keywords: Embedded databases, Tamper-resistant databases, Smart card, Access control, Storage and indexation models, Query evaluation.

[1] SMIS Project – Nicolas.Anciaux@inria.fr
[2] SMIS Project – Luc.Bouganim@inria.fr
[3] SMIS Project – Philippe.Pucheral@inria.fr

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 Le Chesnay Cedex (France)
Téléphone : +33 1 39 63 55 11 — Télécopie : +33 1 39 63 53 30

SGBD embarqué dans une puce : retour d'expérience

Résumé: La carte à puce est aujourd'hui l'objet portable sécurisé le plus répandu. Il y a 4 ans, nous avons jeté les bases d'une étude portant sur l'embarquement de techniques bases de données dans une carte à puce. Cette étude a conduit à la définition de principes de conception pour ce que nous avons appelé alors PicoDBMS, un système de gestion de bases de données (SGBD) complet intégré dans une carte à puce. Depuis, grâce au progrès du matériel et aux efforts conjoints de notre équipe et de notre partenaire industriel, les principes définis initialement ont donné naissance à un prototype complet tournant sur une plate-forme carte à puce expérimentale. Cet article reconsidère la formulation du problème initial à la lumière des évolutions matérielles et applicatives. Il introduit ensuite un banc d'essai dédié aux bases de données embarquées dans des puces et présente une analyse de performance détaillée de notre prototype. Enfin, il dresse des perspectives de recherche dans le domaine de la gestion de données dans les puces sécurisées.

Mots clés: Bases de données embarquées, Confidentialité et protection des données, Carte à puce, Contrôle d'accès, Modèles de stockage et d'indexation, Evaluation de requêtes.

1 Introduction

Thirty years ago, Moreno's patent marked the genesis of the prestigious smart card story, making it today the most widespread secured portable computing device. Smart cards have been used successfully around the world in various applications such as banking, pay-TV or GSM subscriber identification, loyalty, healthcare and transportation. Standards for multi-application support, like JavaCard [1], revolutionized the development and distribution of smart card applications. As smart cards became more and more versatile, multi-application and powerful (32 bit processor, more than 1MB of stable storage), the need for database techniques arose. Basically, embedding query processing, access rights and transaction management in the card makes application code smaller and safer. However, smart cards have specific hardware constraints (tiny RAM, very costly write in stable storage…) making database techniques employed in traditional DBMS and even lightweight DBMS [2][3][4][5] irrelevant. In a previous paper, we addressed the problem of scaling down database techniques for the smart card and proposed the design of what we called a PicoDBMS [6]. This preliminary work opened up exciting research perspectives towards the design of DBMS kernels dedicated to new forms of ultra-light computing devices.

Four years after the publication of this design, three important questions deserve to be raised. The first question is undoubtedly "Is scaling down database techniques for smart cards still a topical issue today?". In other words, the question is whether the hardware progress and the applications evolution drastically change the initial problem statement. "Can the PicoDBMS design effectively accommodate real hardware platforms?" is also an important question related to the feasibility of the approach and introducing a valuable discussion about lessons learned at experimentation time. Finally, "Is PicoDBMS the definite answer to a sporadic problem or rather a first step towards a broader and longer term research?" is a legitimate question putting database components on secured chip in perspective. The objective of this paper is precisely to answer these three questions.

Regarding the evolution of the problem statement, smart card hardware resources do not escape Moore's law in terms of CPU speed and memory size. However, the problem stated in [6] puts more emphasis on the resource dissymmetry (e.g., powerful CPU vs. tiny RAM, very fast read vs. very low write in stable storage, etc) than on their scarcity. This balance did not evolve much is the recent past and, according to smart card manufacturers, should remain rather stable in the near future. It is anyway mandatory to assess the precise impact of hardware progress on the initial PicoDBMS design. Application requirements suffered recently a stronger mutation. The PicoDBMS design was mainly motivated by the management of secured portable folders with an emphasis on healthcare folders. In the last years, secure portable folders have received a growing attention from major industrial actors, broadening their scope of application (e.g., MasterCard's Open Data Store [7]). In addition, the introduction of secured chips in usual computing infrastructures is paving the way for new large scale applications. First, secured chips are being integrated in PC platforms [8] and consumer appliances [9] to prevent piracy and to support advanced Digital Right Management models [10][11] where conditions can be evaluated on consumer's profile and/or historical data. Hence, these data need to be protected. Second, ambient intelligence is flooding many aspects of our everyday life with smart objects gathering information about our habits and preferences, threatening the citizen's elementary rights to privacy. Therefore, the need for secured portable folders protecting the privacy of personal data evolves towards secured (fixed or portable) folders protecting the confidentiality and integrity of either personal or external data. The first contribution of this paper is to revisit the PicoDBMS problem statement in the light of the hardware and application evolutions.

A full-fledged PicoDBMS prototype was initially developed in JavaCard and demonstrated on a Palmera smart card, a commercial platform provided by Axalto (the Schlumberger smart card subsidiary) [12]. While this prototype established the feasibility of the approach, its per-

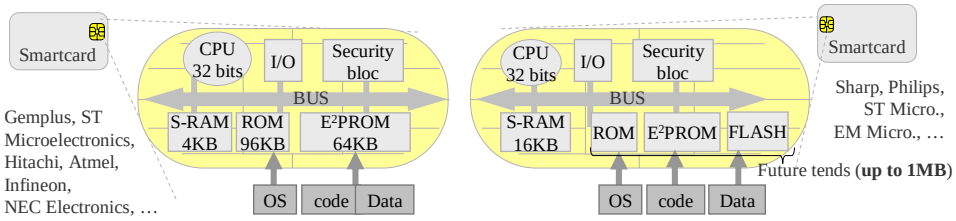
formance was actually far from those expected in [6]. This prototype has been rewritten in C, polished, and recently adapted to ZePlatform, the future generation of Axalto's smart card platform. Three years were necessary to get this new hardware platform and to adapt its OS kernel (with the help of Axalto) to meet the performance requirements of data intensive applications. A benchmark dedicated to Pico-style databases has been designed and used to assess the relative performance of candidate storage and indexation data structures. In addition, a hardware cycle-accurate simulator has been used to predict the performance of our PicoDBMS engine on databases that largely exceed the current smart card storage capacity. The second contribution of this paper is to expose the valuable learning obtained during this long and tricky experimentation process.

If PicoDBMS has proved to be an effective answer to the initial problem statement (i.e., secured portable folder in smart cards), this does not mean however that this problem statement encompasses all the situations where data management can benefit from secured chips. To illustrate this, chip manufacturers are announcing in the short term Mass Storage Cards[4] combining a secured chip with a huge amount (several GB) of unsecured stable memory. Hence, the amount of data to be considered and the new forms of attacks to be faced impose to strongly revisit the database technology on chip. Contactless environments, future (long-term) stable storage technologies, co-design requirements are other considerations deserving research efforts. The third contribution of this paper is therefore to sketch important research directions related to data management issues on secured chips.

The paper is organized as follows. Section 2 revisits the PicoDBMS problem statement according to the evolution of smart cards technologies and usages, answering the first question raised in the introduction, and presents the related works. Section 3 recalls from [6] important aspects of the PicoDBMS internals (i.e., storage model and query processing). This technical description gives the pre-requisites to the in depth performance analysis developed in Section 4. Section 4 first introduces the objectives of the experimentation, exposes the chosen metrics, then discusses the code/data footprint and the query execution performance for each considered storage structure. This section answers the second question raised in the introduction. The third question is actually the subject of Section 5, which draws important research perspectives we foresee in the context of data management on secured chips. Finally, Section 6 concludes the paper.

2 DBMS on Chip: Evolution of the Problem Statement

2.1 Evolution of Smart Card Technologies



(a) Current smart cards (b) Future (short-term) smart cards

Figure 1. Typical smart cards architectures.

Current powerful smart cards include in a monolithic chip, a 32 bit RISC processor cadenced at about 50 MHz, memory modules composed of about 96 KB of ROM, 4 KB of static RAM and 64 KB of EEPROM, and security modules (random number generator, cryptographic co-processor, etc.). The ROM is used to store the operating system, fixed data and standard rou-

[4] MOPASS CONSORTIUM. http://www.mopass.info/english/

tines. The RAM is used as working memory (heap and stack). EEPROM is used to store persistent information, and holds data and downloaded programs (in case of multi-application cards). Figure 1(a) pictures this typical smart card architecture.

The smart card internal resource balance is very unique. Indeed, the on-board processing power, calibrated to sustain cryptographic computations and enforce smart cards security properties, is oversized wrt the small amount of embedded data. In addition, stable memory (EEPROM) shares strong commonality with RAM in terms of granularity (direct access to any memory word) and read performance (60-100 ns/word), but suffers from a dramatically slow write time (about 10 ms per word). Finally, only a few bytes of RAM remain available for the embedded applications, the major part of the RAM being preempted by the operating system and the virtual machine (in case of Java enabled cards). This unusual computing environment – compared to traditional servers running DBMSs – can be summarized by the following properties:

(P1) Very high processing power wrt the small embedded data amount.

(P2) Very tight RAM wrt the amount of data to be processed.

(P3) Fast reads and slow writes in stable memory.

The first PicoDBMS design was driven by these properties. It is therefore important to assess to which extent these properties are impacted by the hardware progress.

2.1.1 Analysis of Property P1

During the last ten years, embedded processors improved from the first 8-bit generation cadenced at 2 MHz to the current 32 bit generation cadenced at 50 MHz with an added cryptographic co-processor. One can consider that existing CPU are powerful enough to meet existing applications requirements and that users might prefer less functionality at a lower cost [14]. On the other hand, we think that at least three factors advocate for a continuous growth of the CPU power. First, the rapid evolution of smart card throughput (e.g., high delivery contactless cards, USB cards) allows smart card applications to evolve towards CPU intensive data flows processing using cryptographic functionalities [15] (e.g., pay TV based on chip decryption and access control over embedded profile). Second, many research efforts and manufacturers investments focus on the development of multi-application and multi-threaded smart card operating systems necessitating even more CPU power. Finally, the smart card cost is mainly driven by the amount of embedded memory (RAM and EEPROM) rather than by the amount of processing power. Smart card being a mass market technology, this consideration favors enhancing the processing power rather than the amount of memory. Hence we are convinced that property P1 will remain valid in the future.

2.1.2 Analysis of Property P2

Currently, smart cards hold a few KB of static RAM, almost entirely consumed by the operating system and by the Java virtual machine in Java cards. Despite the evolution of the computing power (that roughly follows Moore's law), the gap between the RAM available to the applications and the other resources (e.g., CPU speed and stable storage capacity) will certainly keep on increasing for two reasons. First, manufacturers tend to reduce the hardware resources to their minimum in order to save production costs on large-scale markets[5]. The cost of a chip being closely linked to its surface, the relative cell size of static RAM (SRAM cell is 16 times less compact than ROM and FLASH, 4 times less compact than EEPROM) makes it a critical component in the overall cost. This leads to calibrate the RAM size to its minimum, according to the applications requirements [16]. Second, there is today no clear motivation to enlarge the RAM area because (i) the smart card is not autonomous (it takes its electrical energy from the reader), thereby precluding some form of asynchrony (e.g., buffered writes would be lost if the

[5] Semiconductors Industrial Association, STATS: SIA Annual Databook, 2002. http//:www.semichips.org

card is withdrawn from the reader); (ii) Read accesses to stable storage are direct and fast making traditional cache hierarchy not mandatory; and (iii) the RAM competes with other components on the same silicon die [17][18]. Thus, the more RAM the less stable storage while the trend is on embedding more and more persistent data. As a consequence, smart card manufacturers privilege stable storage to the detriment of RAM (see announced smart cards products in table 1). RAM is strictly calibrated to hold the execution stack required by the on-board programs and the RAM increase is only motivated by the support of more complex operating systems and virtual machines.

Table 1. Short-term smart card memory capacity (in KB).

Manufacturer	*Product name*	*ROM*	*FLASH(**)*	*EEPROM(*)*	*RAM*
Renesas	AE57C[6]	368	-	132	10
ST Micro.	ST22N256[7]	368	-	256	16
ST Micro.	ST22FJ1M[8]	128	768	256	16
Philips	P9ST024[9]	512	256	256	16

***** **Refers to traditional EEPROM and to ST Page-FLASH (have the same read/write access time).**
****** **Refers to coarse-grain FLASH usually used to store application code.**

2.1.3 Analysis of Property P3

Let us now consider how hardware advances can impact the memory size limit, and its read/write performance asymmetry. Current smart cards rely on a well established and slightly out-of-date hardware technology (0.35 micron) to minimize production costs and increase security [19]. However, the market pressure generated by emerging applications lead to a rapid increase of the smart card storage capacity. Taking advantage of a 0.18 micron technology allows embedding up to 256 KB of EEPROM (see table 1). In the same time, several manufacturers are attempting to integrate denser memory on chip. Current prototypes integrate FLASH memory for its high compactness (as compact as ROM, much more than EEPROM). However, while read and write times in FLASH are roughly comparable to EEPROM, a whole FLASH bank need to be erased before writing it, making the total cost of an update very high. This makes FLASH memory inappropriate for data-centric applications managing usually a large amount of small data. Thus, the trend is combining a traditional FLASH holding the programs with an EEPROM used as the data store [20]. In a long term perspective, researchers in memory technologies aim at developing the perfect alternative, meaning a highly compact non-volatile memory providing fast read/write operations with fine grain access (e.g., MEMS, OUM, PCM, etc.). However, the integration of such technology in the smart card context is an even longer perspective due to intrinsic difficulties (very high security level that need to be proved for any new technology, low manufacturing cost motivating the usage of old amortized technologies, complexity to integrate all components in the same silicon die).

2.1.4 Conclusion

In the light of the preceding analysis, and despite the undeniable hardware progress, we consider that properties P1, P2 and P3 are still valid today and will remain valid in the short-term (see figure 1(b)) and medium-term. Future memory technologies will probably modify this assumption but must be regarded as a (very) long term perspective.

[6] See http://www.electronicstalk.com/news/ren/ren117.html for a press release.
[7] See http://www.st.com/stonline/books/pdf/docs/10705.pdf for the documentation.
[8] See http://www.st.com/stonline/books/pdf/docs/10497.pdf for the documentation.
[9] See http://www.semiconductors.philips.com/markets/identification/products/hipersmart/ for a description.

2.2 Evolution of Smart Card Usage

The first smart card application was developed for the French banking system in the 1980s to reduce the losses associated with magnetic credit card fraud. Since then, smart cards have been used successfully in a wide range of applications like GSM subscriber identification, pay TV [15], electronic wallet [21], access to network resources, insurance and healthcare [22]. Now, large-scale governmental projects are pushing for an even wider acceptance of smart cards among applications, states and countries. For example, the NETC@RDS Project [23] aims at improving the access of mobile European citizens to national health care systems. The US Federal Government Smart Card Interoperability Specification [24] aims at using smart cards as a vector for employees to access building, to secure networks, but also to digitally sign or encrypt documents such as secure e-mail and administrative requests. Also, multi-purpose ID cards projects [25][26] (e.g., passport, driving license, e-voting, insurance, transport, etc.) have been launched in many countries, including Europe [27], USA [28][29][30], Japan [31][32] and China [33]. All these applications are mainly concerned with user identity control but each application comes with a set of persistent data to be managed and protected.

The form factor of smart cards also evolved along the years to cope with new usages. This gave birth to a wide range of secured chips like ibuttons [34], rings (e.g., JavaRing), serial/parallel/USB secure dongles, and USB "crypto tokens". Most of these devices share strong commonalities with traditional smart cards since they use an embedded 32 bit microcontroller to manage the security process and simply differ in their interface to the host they connect to [35]. Secured chips are sometimes integrated at the hearth of a computing system like in the TCPA architecture to protect PC platforms against piracy [8] or in the SmartRight architecture to enforce Digital Right Management at the rendering devices [9]. Finally, chips are today integrated in a large diversity of usual objects – making them smarter – to form an ambient intelligence surrounding. While security seems not the primary concern in this latter case, the strong demand of individuals for enforcing their elementary rights to privacy may quickly change the situation. As a conclusion, smart card-like secured chips should be almost everywhere in the very short term. The question now is whether data management techniques need to be embedded on these devices. To help answering this question, we introduce below four representative application scenarios.

Healthcare folder: healthcare folder is representative of portable personal folders exhibiting strong database requirements and has been the motivating example of the PicoDBMS study. The information stored in the future health cards should include the holder's identification, insurance data, emergency data, the holder's doctors, prescriptions and even links to heavier data (X-ray examination…) stored on hospital servers. Different users may share data in the holder's folder with different privileges: the doctors who consult the patient's past records and prescribe drugs, the pharmacists who deliver drugs, the insurance agents who refund the patient, public organizations which maintain statistics or study the impact of drugs correlation in population samples and the holder herself. HIPAA specifications [36], which edicts principles protecting personal healthcare information, make smart cards an ideal partner to hold medical data [37][38]. Embedded data management techniques are mandatory to actually store, organize and query this huge amount of data and to control the respective privileges of each user.

Customer wallet: MasterCard recently introduced the MasterCard Open Data Storage (MODS) specifications [7] to enable cardholders and service providers (or any merchant third party) to store and retrieve objects directly on customers' smart cards. The benefit for service providers is to keep aware of the consumer habits. The main benefit for the customer is to take advantage of special offers related to her past consumption. However, the price to pay for the latter benefit is high: not only the customer activity can be monitored in each shop the customer visits, but also the cross references between her different activities. The enforcement by the smart card of sophisticated access rights are therefore a prerequisite to give the control on the stored data back to the customer. As stated by MasterCard itself, the MODS API must "*meet the desire expressed by customers to better control how much information they are willing to*

share with whom" [7]. The IBM-Harris report on consumer privacy survey strongly highlights this same requirement [39].

Digital Right Management (DRM): Digital piracy is threatening the global multimedia content industry (a 16% decline of the music market revenues since 1999 [40]) while forecast institutes agree on the unprecedented potential of the mobile audio and video market. Basic DRM models fail in solving this paradox because they badly adapt to several new attractive usage scenarios and because consumers are reluctant to use them for privacy preservation and fairness concerns. Several initiatives [10][41][11] demonstrate the need for expressive DRM languages. Such languages allow business rules to express conditions on historical data, on user's profile and on contextual information (e.g., Alice may listen freely to a piece of music according to her past activity or Bob may download a given e-lesson provided he is registered as student in this course and passed the preceding exam successfully). These data must be protected against information leakage injuring privacy as well as against tampering from the user herself. Smart card data management techniques are well suited to answer this requirement [42].

Aware Home: smart objects are invading the domestic area to capture the habits and preferences of individual with the laudable objective to provide them with a personalized service. Once organized and centralized, such personal data are however rather sensible and need to be protected against potential misuse. In another context, Hippocratic databases [43] have been introduced as a mean for the data owner to control the usage of her data by giving (or refusing) her consent to the exploitation of a given data for a given purpose under given conditions. We advocate here the use of secured chip embedding data storage and access right management to implement Hippocratic smart objects.

These four scenarios do nothing but illustrating the same needs for data management techniques on chip in different contexts.

2.3 Problem Statement

The initial formulation of the PicoDBMS problem [6] was strongly influenced by the personal medical folder example (the best motivating example at that time). It stated the necessity to embed on a smart card the data to be protected as well as the software components required to manage them, namely a Storage Manager, a Query Manager, a Transaction Manager and an Access right manager. Then it introduced seven rules to design these components according to the smart card constraints.

The approach is still valid and we will follow the previous paper footstep to express the problem statement. However, the baseline is slightly different. As discussed in the preceding section, several scenarios can serve today as motivating examples and the PicoDBMS design must satisfy all. While the form of the data, the expected transactional properties and the query facilities requirements may vary among the scenarios, they all share the same ultimate requirement for a powerful access control management enforcing the confidentiality of embedded data. This point strongly distinguishes a PicoDBMS from a traditional DBMS. Querying the data should even not be a concern for a PicoDBMS. What is actually expected from a PicoDBMS is to store the data securely (thanks to the smart card tamper resistance) and to act as a trusted doorkeeper, which authenticates each user and solely delivers her authorized view of the data. The computation made on these data after their delivery is not the matter.

In the light of this remark, the main point is to determine which data management techniques are required by this doorkeeper. We make the assumption that the user identification/authentication is ensured traditionally (e.g., by a login/password) and does not deserve further discussion. PicoDBMS has been designed in the relational context. Hence, the considered access control model is the SQL one. For the sake of simplicity, we concentrate on the core of the model and consider that privileges are granted/revoked to users on relations or views. Since data confidentiality is the primary concern, we introduce below a classification of the read access authorizations a PicoDBMS must be able to express and enforce.

Table 2. Description of access rights on embedded data.

Acronym	Auth°	Access right type	Required database operator(s)	Example[10]
P	SA	Subset of attributes	Project	Name of doctors
SP	OA	Predicates on a single relation	Select, Project	Name of non-specialist doctors
SPJ	OA	Predicates on two relations (direct relationship)	Select, Project, Join	Name of doctors the patient has visited last month
SPJ^n	OA	Predicates on several relations (transitive relationship)	Select, Project, Joins	Name of doctors who prescribed antibiotics to the patient
SPG	CA	Single relation, mono-attribute aggregation	Select, Project, Group (aggregate)	Number of doctor per specialty
SPJG	CA	Several relations, mono-attribute aggregation	Select, Project, Join, Group (aggregate)	Number of prescriptions per doctor's specialty
$SPJG^n$	CA	Several relations, multi-attribute aggregation	Select, Project, Join, Group (aggregate)	Number of visits per doctor's specialty and per type of drug

For the sake of generality, the categories of authorizations are expressed over an entity relationship database schema. Schema authorizations (SA) are defined on the database intension (schema) without considering the database extension (occurrences). Occurrence authorizations (OA) grant access to some occurrences, depending on predicate(s) expressed on their properties, or on the existence of a relationship (either direct or transitive) with another granted occurrence. Computed data authorizations (CA) grant access to computed values without granting access to the occurrences taking part in the computation (e.g., averaged values are disclosed but not the related individual records).

We distinguish below seven representative authorization types for a relational database, derived from the SA, OA and CA classes. This classification is important to determine the expected functionalities of a PicoDBMS and will serve as well to assess the performance of the system. table 2 summarizes these authorization types. SA authorizations may be implemented by views involving the project operator (P access right). The predicates of OA can apply to attributes of one relation (SP access right), two relations (direct relationship, SPJ access right), or several relations (transitive relationship, SPJ^n access right). Finally, CA are implemented by means of views involving aggregates. The aggregation may be mono-attribute (i.e., group by on a single attribute) and consider a single relation (SPG access right), mono-attribute and consider several relations (SPJG access right), or multi-attribute, potentially considering several relations ($SPJG^n$ access right).

A trivial solution to enforce these authorizations would be to materialize each view in a separate file and to externalize this file when a user requests it. This undoubtedly would reduce the PicoDBMS footprint to its minimum (file management) and ease its integration in the smart card. However, this badly adapts to data and authorization updates and would result in a huge data replication among files, thereby hurting the smart card limited storage constraint.

Thus, a dynamic evaluation of the authorized views must be considered as a prerequisite in the problem statement. This implies to embed on chip the ability to compute database operators and run query execution plans implementing views. The problem statement can thus be formulated as follows:

1. to design a DBMS supporting SA, OA, and CA authorizations;
2. to support accurately data updates and access control policies evolutions;
3. to comply with the strong smart card hardware constraints.

This problem statement leads to a set of design rules derived from the smart card properties:

[10] The examples show that the complexity of authorization is not related to the size of a database. A medical folder database – composed of the Doctor, Visit, Prescription and Drug relations – is considered but similar examples could be derived from all application scenarios mentioned above.

- *Compactness rule*: minimize the size of the data, indexation structures and PicoDBMS footprint to cope with the limited stable storage area.
- *RAM rule*: minimize the RAM consumption of all operators given its extremely limited size.
- *Write rule*: minimize write operations given their dramatic cost (≈10 ms/word).
- *Read rule*: take advantage of the fast read operations in stable storage (≈100 ns/word).
- *Access rule*: take advantage of the low granularity and direct access capability of the stable memory for both read and write operations.
- *Security rule*: never externalize private data from the chip and minimize the algorithms complexity to avoid security holes.
- *CPU rule*: take advantage of the over-dimensioned CPU power, compared to the amount of embedded data.

2.4 Related Works

From the early 90's, the need for data management appears in several applications running on small devices, from industrial controllers and cable-television set-top boxes to medical-imaging systems and airplanes flight controls. This motivates manufacturers of embedded solutions to focus on the development of embedded DBMSs. The objective of embedded DBMSs is to provide efficient data access [44][45] and transactional features (ACID properties), while cohabiting with others applications in the device (especially the ones using the database). The embedded DBMS design has been driven so far by code footprint minimization, obtained by simplification and componentization, portability on multiple devices / operating systems, and self-administration [46]. Generally, embedded DBMS are dynamically linked with the embedded application. Few academic studies have been conducted in this area [47][48], but many commercial products exist like *Pervasive SQL 2000* [49], *Empress database* [50], and *Berkeley DB*[51].

In parallel, the rapidly growing number of mobile phones, PDAs and other portable consumer devices stimulates the main DBMS editors to adapt their products to this promising market [4]. Thus, light versions of popular DBMS like *IBM DB2 Everyplace* [52][3], *Oracle Lite* [2], *Sybase SQL Anywhere Studio* [5], *Microsoft SQL Server for Windows CE* [4] have been designed for small devices. The DBMS editors pay a great attention to data replication and synchronization with a central database in order to allow offline executions and resynchronization at reconnection time. Again, the small footprint is obtained by simplifying the DBMS code and by a selective linking of the DBMS code with the application code. Light DBMSs are portable to a large variety of small devices and operating systems (*e.g.*, *PalmOs*, *WinCE*, *Embedded Linux*). While addressing different market shares and having followed different development paths, it is now quite difficult to clearly distinguish embedded DBMSs from light DBMSs. Actually, the former approach adds incrementally DBMS features (transaction support, SQL management, etc.) while the latter removes unnecessary features to minimize the code footprint.

By considering smart cards as traditional small devices, one could envision using embedded DBMSs or Light DBMSs technology in this environment. However the primary objective of smart cards being security, smart cards have a very specific hardware architecture. The hardware resource asymmetry pointed out in Section 2.3 entails a thorough re-thinking of the techniques usually used for embedded / light DBMSs.

Sensor networks gathering weather, pollution or traffic information have motivated several recent works related to on-chip data management. *Directed Diffusion* [53], *TinyDB* [54], and *Cougar* [55] focus on the way to collect and process continuous streams of sensed data. Optimization techniques for queries distributed among sensors are investigated in [56][57]. Techniques to reduce power consumption induced by radio frequency communications, one of the major sensor constraints, are proposed in [58]. Although sensors and smart cards share some hardware constraints, the objectives of both environments diverge as well as the techniques

used to meet their respective requirements. Basically, the on-chip database capabilities required in a sensor are limited to data filtering and aggregation to reduce the output flow of the sensor thereby saving energy [59]. In addition, response-time constraints are strongly different in both cases. Depending on the usage of the sensor network, time constraints may be either inexistent or hard real-time while time constraints in the smart card context are related to the user interaction. Finally, sensors often compute queries on streaming data or on a single local embedded relation while a smart card PicoDBMS may have to manage several relations.

A small number of studies have specifically addressed the area of data management on smart cards. The first attempts towards a smart card DBMS were ISOL's SQLJava Machine and the ISO standard for smart card database language, SCQL [60]. Both were addressing generation of smart cards endowed with 8 KB of stable memory. While SQLJava Machine and SCQL designs were limited to mono-relation queries, they exemplify the growing interest for smart card DBMS. More recently, the MasterCard Open Data Storage initiative (MODS) [7] promoted a common API to allow retailers, banks and other organizations to access and store data on users' smart cards with an enhanced security for the smart card holder. However, MODS is based on flat files, crude access rights (*i.e.*, file level access rights), and thus, does not provide neither compact data model nor query processing. Again, this initiative shows the interest of developing smart card DBMS techniques.

A recent study proposes specific storage techniques to manage data in flash memory on a smart card [61]. The design is also limited to mono-relation queries and is strongly impacted by the physical characteristics of the target smart card architecture. Indeed, they propose to store the data in NOR type of FLASH memory generally dedicated to store programs as ROM replacement. Since updates in NOR flash memory are very costly (updating a single data induces a large and costly bloc erasure), the techniques are driven by update cost minimization (using dummy records and deleted bits). While this study shows the impact of hardware characteristics on the DBMS internals, it does not comply with the memory constraints of the smart card nor addresses complex query processing, mandatory to extract the authorized part of the data.

Finally, let us remark that *GnatDB* [62], presented as a "small-footprint, secure database system" may fit in a smart card since it has an 11 KB footprint. Actually, GnatDB was designed to manage a small amount of external data, the smart card being used to protect remote data against accidental or malicious corruption. GnatDB thus does not address query processing nor internal data storage and indexation. As a conclusion, PicoDBMS [6] was the first study considering a full-fledged DBMS on smart card.

3 PicoDBMS at a Glance

As the problem statement makes clear, the central point of the study is on evaluating dynamically authorized views implementing SA, OA, and CA authorizations on chip. We recall from [6] the basic principles of PicoDBMS that are related to this issue. We do not discuss others aspects, except when they infer in the evaluation process (e.g., transaction processing), in which case we give the minimum information required for the understanding.

The modules that must be present on-chip wrt the confidentiality target are: a storage manager organizing data and indices into the chip stable memory, a query manager building execution plans (including selections, projections, joins and aggregates operators) and capable of processing rather complex views on-chip and finally an access right manager providing grant and revoke functionalities on the defined user views and preventing from externalizing unauthorized data. A transaction manager enforcing the atomicity of a sequence of updates must also be embedded on chip for consistency purpose.

The other modules do not impact data confidentiality. Thus, they can be stored on the terminal, within the JDBC driver interfacing the user application with the embedded database engine. User's authorized database schema is externalized at connection time, allowing the JDBC to parse the query and verify the SQL syntax. As well, the order by clause, if any, is evaluated on the terminal. Note that these remote computations do not hurt the security rule since they

only involve authorized information. Thus, PicoDBMS allows confidential data sharing among several users connecting to the database one at a time. Figure 2 pictures the PicoDBMS architecture.

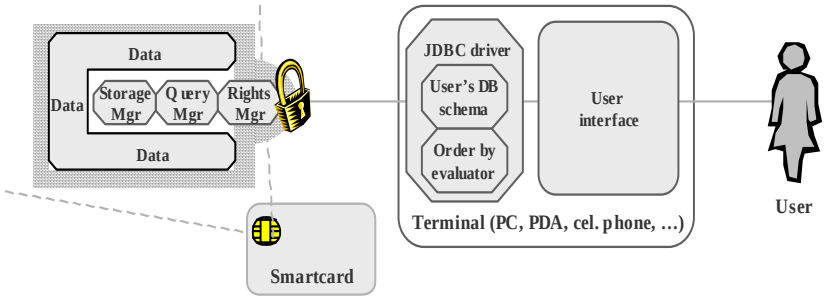
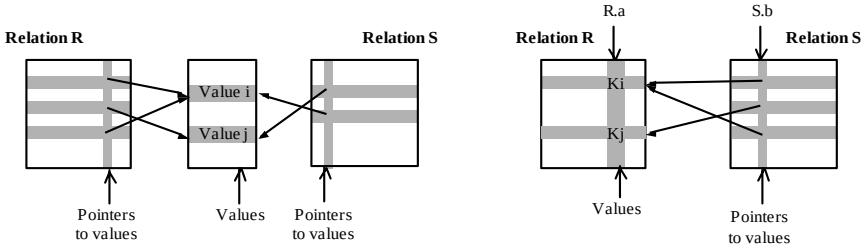


Figure 2. PicoDBMS Architecture.

3.1 Storage and Indexation

The simplest way to organize data is Flat Storage (FS), where tuples are stored sequentially and attribute values are embedded in the tuples. The main advantage of FS is access locality. However, in our context, FS has two main drawbacks. First, this storage organization is space consuming. While normalization rules preclude attributes conjunction redundancy to occur, they do not avoid attribute value duplicates. Second, it is inefficient, inducing the sequential computation of all operations in the absence of index structures. Adding index structures to FS may solve the second problem while worsening the first one.



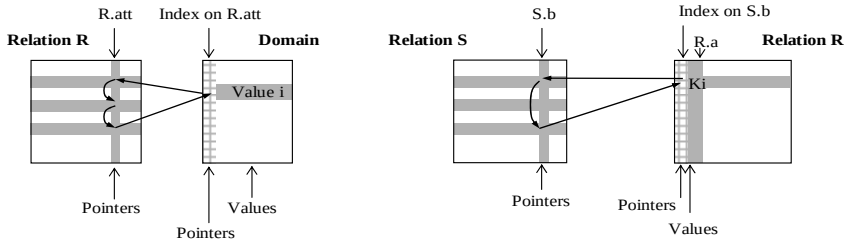
(a) Domain Storage of a Regular Attribute. (b) Domain Storage of Foreign-key Attribute.

Figure 3. Domain Storage.

Based on the critique of FS, it follows that a PicoDBMS storage model should guarantee both data and index compactness. Since locality is no longer an issue in our context (read rule), pointer-based storage models inspired by MMDBMS [63][64][65] can help providing compactness. The basic idea is to preclude any duplicate value to occur. This can be achieved by grouping values in domains (sets of unique values). We call this model Domain Storage (DS). As shown in figure 3, tuples reference their attribute values by means of pointers. Furthermore, a domain can be shared among several attributes. This is particularly efficient for enumerated types, which vary on a small and determined set of values.

One may wonder about the cost of tuple creation, update and deletion since they may generate insertion and deletion of values in domains. While these actions are more complex than their FS counterpart, their implementation remains more efficient in the smart card context, simply because the amount of data to be written is much smaller (write rule). To amortize the slight overhead of domain storage, we only store by domain all large attributes (i.e., greater than a pointer size) containing duplicates. Obviously, attributes with no duplicates (e.g., keys) need not be stored by domain but with FS. Variable-size attributes – generally larger than a pointer – can

also be advantageously stored in domains even if they do not contain duplicates. The benefit is not storage savings but memory management simplicity (all tuples of all relations become fixed-size) and log compactness.



(a) Ring Index on a Regular Attribute. (b) Ring Index on a Foreign-key Attribute.

Figure 4. Ring Storage.

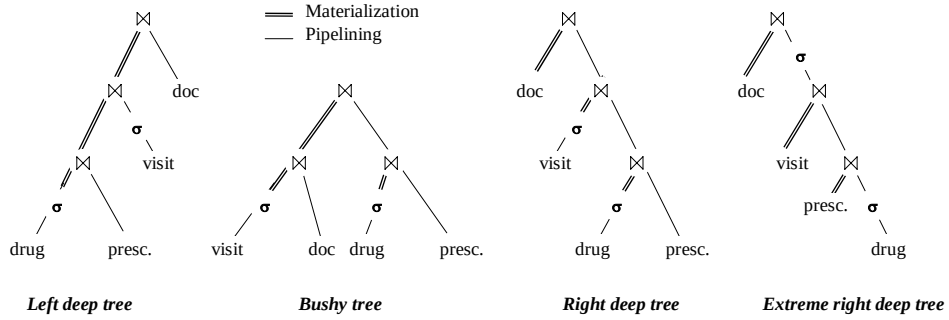
We now address index compactness along with data compactness. Unlike disk-based DBMS that favor indices which preserve access locality, smart cards should make intensive use of secondary (i.e., pointer-based) indices. The point here is to make these indices efficient and as compact as possible, integrating them in the storage model instead of additional structures. Let us first consider select indices. A select index is typically made of two parts: a collection of values and a collection of pointers linking each value to all tuples sharing it. Assuming the indexed attribute varies on a domain, the index collection of values can be saved since it exactly corresponds to the domain extension. The extra cost incurred by the index is then reduced to the pointers linking index values to tuples. Let us go one step further and get these pointers almost for free. The idea is to store these value-to-tuple pointers in place of the tuple-to-value pointers within the tuples (i.e., pointers stored in the tuples to reference their attribute values in the domains). This yields to an index structure which makes a ring from the domain values to the tuples, as shown in figure 4(a). The ring index can also be used to access the domain values from the tuples and thus serve as data storage model. Thus we call Ring Storage (RS) the storage of a domain-based attribute indexed by a ring. The index storage cost is reduced to its lowest bound, that is, one pointer per domain value, whatever the cardinality of the indexed relation. This important storage saving is obtained at the price of extra work for projecting a tuple to the corresponding attribute since retrieving the value of a ring stored attribute means traversing in average half of the ring (i.e., up to reach the domain value).

Join indices [66] can be treated in a similar way. A join predicate of the form (R.a=S.b) assumes that R.a and S.b vary on the same domain. Storing both R.a and S.b by means of rings leads to define a join index. In this way, each domain value is linked by two separate rings to all tuples from R and S sharing the same join attribute value. As most joins are performed on key attributes, R.a being a primary key and S.b being the foreign key referencing R.a, key attributes are stored with FS in our model. Nevertheless, since R.a is the primary key of R, its extension forms precisely a domain, even if not stored outside of R. Since attribute S.b takes it value in R.a's domain, it references R.a values by means of pointers. Thus, the domain-based storage model naturally implements for free a unidirectional join index from S.b to R.a (see figure 4(b)). If traversals from R.a to S.b need be optimized too, a bi-directional join index is required. This can be simply achieved by defining a ring index on S.b. Figure 4(b) shows the resulting situation where each R tuple is linked by a ring to all S tuples matching with it and vice-versa. The cost of a bi-directional join index is restricted to a single pointer per R tuple, whatever the cardinality of S. This index structure can even be enhanced by combining RS and DS. This leads to a bi-directional join index providing direct (i.e., one single pointer) traversals from S.b to R.a. This enhanced storage model, called Ring Inverse Storage (RIS), requires one additional pointer per S and R tuples. Its storage cost is much more expensive, and thus must give a high performance benefit to be adopted.

3.2 Query Processing

Traditional query processing strives to exploit large main memory for storing temporary data structures (e.g., hash tables) and intermediate results and resort to materialization on disk in case of memory overflow. These algorithms cannot be used for a PicoDBMS due to the write rule. To tackle this issue, we propose query processing techniques that do not use any working RAM area nor incur any writes in stable memory.

Let us first consider the execution of SPJ (Select/Project/Join) queries. As usual, the query optimizer first generates an "optimal" query execution plan (QEP) that is then executed. The optimizer can consider different shapes of QEP: left-deep, right-deep or bushy trees. In a left-deep tree, operators are executed sequentially and each intermediate result is materialized. Right-deep trees execute operators in a pipeline fashion, thus avoiding intermediate result materialization, but all left relations need to be materialized. Bushy trees offer opportunities to deal with the size of intermediate results and memory consumption [67].



Healthcare Database schema:(sample)
*Doctor (*DocId, name, specialty, *…)*
*Visit (*VisId, DocId, date, diagnostic, *…)*
*Prescription (*VisId, DrugId, qty, *…)*
*Drug (*DrugId, name, type, *…)*

Query: Name of doctors who prescribed antibiotics after 01/01/2000

```
SELECT D.name
FROM Doctor D, Visit V, Prescription P, Drug d
WHERE D.DocId = V. DocId AND V.VisId = P.VisId
      AND P.DrugId = D.DrugId AND D.type = 'ANTIBIOTIC'
      AND V.date > 01-jan-2000;
```

Figure 5. Several Execution Trees for Query Q1: 'Who prescribed antibiotics in 2000?'.

In a PicoDBMS, the query optimizer should not consider any of these execution trees as they incur materialization. The solution is to only use pipelining with extreme right-deep trees where all the operators (including select) are pipelined. As left operands are always base relations, they are already materialized in stable memory, thus allowing executing a plan with no RAM consumption. Pipeline execution can be easily achieved using the well known Iterator Model [68]. A QEP is activated starting at the root of the operator tree. The dataflow in the model is demand-driven: a child operator passes a tuple to its parent node in response to a *next* call from the parent.

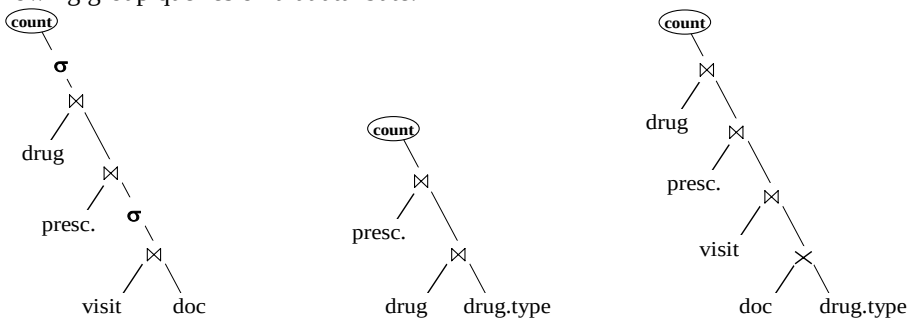
The Select operator tests each incoming tuple against the selection predicates. The values of the incoming tuples needed to evaluate the predicate are directly read (FS) in the tuple and/or reached by dereferencing a pointer (DS/RIS) and/or by following a pointers ring. While indexed (RS storage of the attribute participating in the selection), the selection predicate (or part of, whether multi-attribute) can be evaluated on the distinct values, and the matching tuples are directly retrieved following the pointers rings.

Project simply builds a result tuple by copying the value (FS) and/or dereferencing the cursors (DS/RIS) and/or following the pointers ring to reach the value (RS) present in the input tuple. The project operator is pushed up to the tree since no materialization occurs. Note that the final project incurs an additional cost in case of ring attributes.

Join implements a Cartesian product between its left and right inputs, since no other join technique can be applied without ad-hoc structures (e.g., hash tables) and/or working area (e.g.,

sorting). Each incoming right tuple induce a complete iteration on the left input to retrieve the matching tuples. In case of indices (DS/RS/RIS), the cost of joins depends on the way indices are traversed. Consider the indexed join between Doctor (n tuples) and Visit (m tuples) on their key attribute. Assuming a unidirectional index, the join cost is proportional to n*m starting with Doctor (i.e., right input is Doctor) and to m starting with Visit (i.e., right input is Visit). Assuming now a bi-directional index, the join cost becomes proportional to n+m starting with Doctor and to $m^2/2n$ starting with Visit (retrieving the doctor associated to each visit incurs traversing half of a ring in average). In the latter case, a naïve nested loop join can be more efficient if the ring cardinality is greater than the target relation cardinality (i.e., when $m>n^2$). In that case, the database designer must clearly choose a unidirectional index between the two relations.

We now consider the execution of the aggregate operator (sort operations is not described since it can be performed on the terminal). At first glance, pipeline execution is not compatible with aggregation, classically performed on materialized intermediate results. We propose a solution to the above problem by exploiting two properties: (i) aggregate can be done in pipeline if the incoming tuples are yet grouped by distinct values and (ii) pipeline operators are order-preserving since they consume (and produce) tuples in the arrival order. Thus, enforcing an adequate consumption order at the leaf of the execution tree allows pipelined aggregation. For instance, the extreme pipeline tree of figure 5 delivers the tuples naturally grouped by Drug.id, thus allowing group queries on that attribute.



Q2: *Number of prescribed antibiotics per doctor in 1999*

Q3: *Number of prescription per type of drug*

Q4: *Number of prescriptions per doctor and type of drug*

Figure 6. 'Complex' Query Execution Plans.

Let us consider now query Q2 of figure 6. As pictured, executing Q2 in pipeline requires rearranging the execution tree so that the relation Doctor is explored first. Since the relation Doctor contains distinct doctors, the tuples arriving to the count operator are naturally grouped by doctors. The case of Q3 is harder. As the data must be grouped by type of drugs rather than by Drug.id, an additional join is required between the relation Drug and the domain Drug.type. Domain values being unique, this join produces the tuples in the adequate order. The case of Q4 is even trickier. The result must be grouped on two attributes (Doctor.id and Drug.type), introducing the need to start the tree with both relations. The solution is to insert a Cartesian product operator at the leaf of the tree in order to produce tuples ordered by Doctor.id and Drug.type. In this particular case, the query response time should be approximately n times greater than the same query without the 'group by' clause, where n is the number of distinct types of drugs.

Regarding query optimization, the extreme right-deep trees context makes it rather straightforward. However, simple heuristics can give attractive enhancements. For example, the selections have to be executed as soon as possible, and the access ordering can favor the join indices traversal in the most favorable sense (only one possibility in case of unidirectional index, i.e., DS, and from the referred key relation to the referent foreign key one in case of bi-directional index, i.e., RS/RIS).

3.3 Transaction Processing

A PicoDBMS must enforce the well-known transactional ACID properties. While being out of the scope of this study, we give a minimal description of the transaction processing to take its cost into account in the performance measurements.

Regarding integrity control, traditional techniques are used and do not deserve any particular comments. Only structural (i.e., uniqueness and referential) constraints are considered in the performance evaluation. Regarding concurrency control, PicoDBMS has been developed on mono-threaded platforms and is used by a single user at a time. Hence, concurrency control is not a concern.

The logging process deserves a deeper explanation. Logging is necessary to ensure local atomicity, global atomicity (in a distributed setting) and durability. As stated in [69], the cost of durability as well as global atomicity can be transferred to the network thanks to an ad-hoc protocol named Unilateral Commit for Mobile. The local atomicity must adapt to an update-in-place model because shadow updates behave badly with a pointer-based storage model. A Write-Ahead Logging protocol (WAL) is followed to allow undo aborted updates. Traditional WAL logs the values of all modified data. RS allows a finer granularity by logging pointers in place of values. The smallest the log records, the cheapest the WAL (write rule). The log record contains the tuple address and the old attribute values, that is a pointer for each RS or DS stored attributes and a regular value for FS stored attributes. Rings come for free in term of logging because they can be regenerated from the tuple records at recovery time. In case of a tuple insertion or deletion, only the tuple address has to be logged thanks to a status bit (i.e., dead or alive) in each tuple header. Insertion and deletion of domain values should be logged as any other updates. This overhead can be avoided by implementing a deferred garbage collector that destroys all domain values no longer referenced by any tuple [6].

4 PERFORMANCE EVALUATION AND LESSONS LEARNED

This section evaluates the performance of PicoDBMS and raises the main lessons learned during our experimentation. First, we introduce the performance metrics of the evaluations. Second, we present the PicoDBMS footprint as well as the database footprint incurred by each candidate storage and indexation model, an important issue wrt the compactness rule. Third, we detail an in-depth query performance evaluation – combining measurements and simulations – to assess the ability of PicoDBMS to support complex authorizations with acceptable performance.

4.1 Performance Metrics

By focusing on transaction throughput, existing DBMS benchmarks (typically the TPC family [70]) are clearly inadequate to assess the performance of a PicoDBMS. Hence, the first issue is capturing the main characteristics of Pico-style databases to fix relevant performance metrics. These characteristics are:

(C1) Low upper bound for the database footprint: from tens of KB up to one MB (cf. Section 2, Table I).

(C2) Append-intensive update profile: Pico-style databases often contain historical data, while updates and deletes are less frequent (cf. Section 2.3).

(C3) Sophisticated access rights: the authorized views may combine projections, selections, joins, and aggregates in their definition (cf. Section 2.3, Table 2).

(C4) Chip viewed as a smart storage medium: users (either human or software) connect to the device to access (according to their access rights) the embedded data, analogously to any other storage device (magnetic disc, USB key, etc.).

The following performance metrics are derived from these characteristics:

Data storage capacity: this metrics is expressed in terms of Ktuples/KB for a representative database. It measures (1) the ability of the DBMS engine to compress the on-board data and (2) the DBMS engine footprint itself which competes with the persistent data[11].

Insertion rate: this metrics is expressed in terms of inserted Ktuples/second. The update and delete costs must be acceptable, but are less significant for the target applications.

Latency: this metrics is expressed in terms of time spent in second to produce the first result tuple for a given execution plan (representing a view).

Transmission rate: this metrics is expressed in terms of result tuples produced per second. This metrics is relevant for applications concerned with the data transmission rate or by the total query execution time (e.g., to sort the complete result before display).

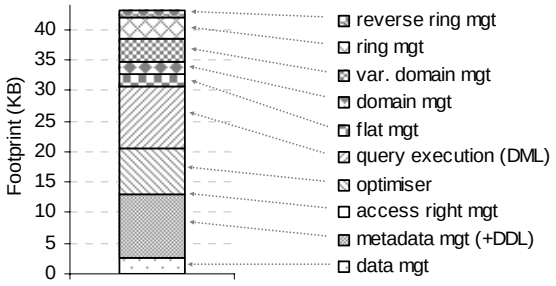
As suggested by characteristic C4, latency and transmission rate are comparable to those employed for evaluating the performance of traditional storage medium. Characteristic C3 imposes measuring these performances for a broad range of queries (i.e., views), representing simple up to rather complex access rights. Finally, note that transaction throughput is not an issue in a PicoDBMS context. The issue is more to guarantee an acceptable response time for a single user/application (a few seconds is often considered as an acceptable threshold for user interactions).

4.2 PicoDBMS Kernel Footprint

Figure 7(b) presents the respective size of the PicoDBMS modules while figure 7(a) gives the total footprint of the main implementation alternatives. While self-explanatory, these figures deserve two remarks.

PicoDBMS version	*Storage model*	*Indexation*	*Footprint (KB)*
Flat-Based	FS	No	26
Domain-Based	FS/DS	Join	30
Ring-Based	FS/DS/RS	Join & select	39
Ring-Inverse-Based	FS/DS/RS/RIS	Join & select	42

(a) Different PicoDBMS Variations.



(b) Ring-Inversed-Based Version of PicoDBMS.

Figure 7. PicoDBMS Code Footprint.

First, the code footprint is often considered as a main challenge and is largely exploited by lightweight DBMS editors for commercial purpose. Our experience contradicts this point. The reason for this is threefold: (1) the gap between the Ring-Inverse-based and the Flat-based versions of PicoDBMS is far less important than expected; (2) ROM cells are 4 times smaller than EEPROM cells and so are the respective benefits of decreasing the code and the data footprint;

[11] In a real setting, the DBMS engine is located in ROM while the data are stored in EEPROM. However, since ROM and EEPROM compete on the same silicon die, both measurements are relevant.

(3) minimizing the code footprint leads to select the flat storage model, strongly increasing the database footprint (see next section).

Second, Figure 7(b) shows that the metadata management module is rather heavy compared to the others. Indeed, several query execution plans cannot fit in RAM simultaneously. This precludes to store metadata inside traditional database relations, and to access it by means of queries. Metadata are therefore stored in ad-hoc structures, and are accessed by calling ad-hoc procedures, explaining the heavy footprint of this component.

4.3 Database Footprint

This section evaluates the database compactness obtained by the four candidate storage and indexation models (FS, DS, RS, and RIS). This evaluation cannot be conducted accurately using a synthetic dataset, unable to translate a real-life distribution of duplicate values. Thanks to our contact with medical partners, we used for this experience a subset of a genuine medical dataset, stored into the simplified database schema presented in figure 8.

Figure 9(a) pictures the data storage capacity for each model and the average tuples per KB for each, e.g. 43 tuples per KB with DS, while figure 9(b) and figure 9(c) plot ratios between these models. RIS performs really badly (RIS is very close to FS) compared to DS and RS. Thus, RIS should be adopted only if it provides a significant performance gain at query execution time. DS and RS provide relatively close results, highlighting the high compactness of ring indices. In addition, the more the embedded tuples, the less the indexation cost and the most effective DS which acts as a dictionary compression scheme. This remark induces that for bigger smart cards, indices comes at a lower cost.

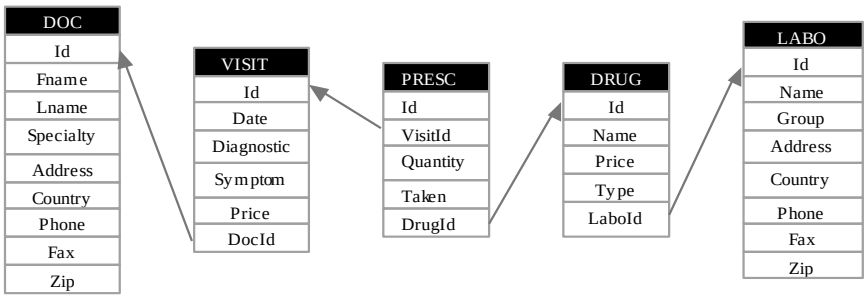
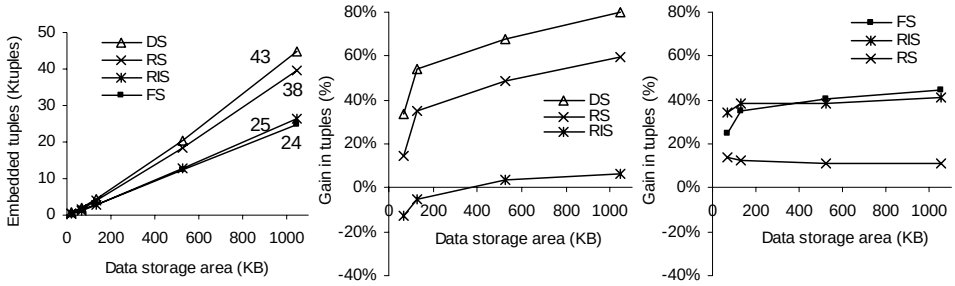


Figure 8. Medical Database Schema.

Note also that the total number of embedded tuples does not exceed 50.000 for a smart card providing 1MB of stable storage. Thus, we do not examine in the sequel queries involving more than this upper bound.



(a) Amount of embedded tuples. (b) Gain Compared to FS. (c) Lost Compared to DS.

Figure 9. Data Storage Capacity.

4.4 PicoDBMS Performance

This section measures the PicoDBMS performance for update (tuple insertion) and queries (view evaluation). Two different platforms have been used to conduct these experiments: a real smart card prototype and a hardware simulator of this prototype (both provided by Axalto), the latter allowing us to consider datasets exceeding the current smart card storage capacity. We present below these two platforms, the dataset and queries considered in our Pico-style benchmark and the performance results.

4.4.1 Experimentation Platforms

The real smart card prototype is equipped with a 32 bit CPU running at 50 MHz, 64 KB of EEPROM, 96 KB of ROM, and 4 KB of RAM (among which only a hundred of bytes remains available for the application). An internal timer allows measuring the response time for each incoming APDU[12]. The PicoDBMS code being located in EEPROM in this prototype, the upper bound for the database footprint is reduced to 22 KB.

To assess the performance on larger datasets (up to 1MB), we used the simulation platform pictured in Figure 10. This platform is made of a hardware simulator of the smart card prototype connected to a control PC, and linked to a client PC connected to a standard smart card reader (ISO 7816 standard). This hardware simulator is cycle accurate, meaning that it monitors the exact number of CPU cycles between two breakpoints set in the embedded code. We checked the accuracy of the simulator comparing the simulator measures on small databases to the one obtained using the real smart card prototype.

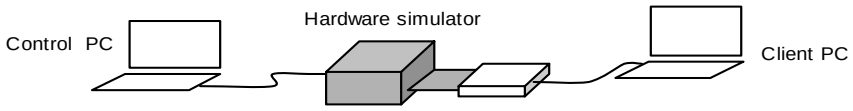


Figure 10. Smart Card Simulation Platform.

The smart card prototype as well as the hardware simulator run a modified version of the Ze-Platform operating system. Indeed, taking advantage of the PicoDBMS experience, Axalto modified the initial version of their operating system to better support data intensive operations (a rather unusual case in traditional smart card applications). Among others, direct accesses to the EEPROM have been optimized, a crucial point for pointer-based storage and indexation models. Thanks to these modifications and to the rewriting of PicoDBMS in C, the performance gain was roughly two orders of magnitude compared with the JavaCard PicoDBMS prototype demonstrated at VLDB'01 [12].

4.4.2 Dataset and Queries

For the performance evaluation, we use a synthetic dataset (thus different from the real dataset used to observe data compactness) to control and vary the selectivity of each attribute. The corresponding database schema is pictured in figure 11. It is representative of a complex embedded database. Relations cardinality ratios are similar to the TPC-R/H/W benchmarks, that is to say a large relation (R0) referencing two relations (R1 and R2) 6 times smaller, each referencing in turn a relation (resp. R3, R4) 6 times smaller.

[12] The APDU (Application Protocol Data Unit) is the communication unit between the smart card and the reader, as defined by the ISO 7816 standard. The communication cost between the smart card and the reader is actually not taken into account in the measurements. Indeed, this cost is not a long term bottleneck, some USB smart cards with a 8Mbps throughput being already developed.

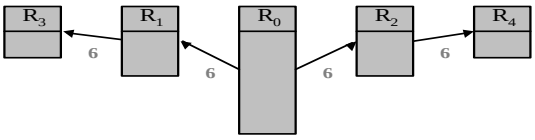


Figure 11. Synthetic Database Schema.

Table 3 details the content of relation R0. The other relations share the same structure, except that they hold a single foreign key attribute. Note that attribute A0 is stored in FS since it is unique and cannot benefit from DS, RS, or RIS models. A3 and A4 are never stored in DS since they have the same size as a pointer and participate only in projections (see below).

Table 3. Schema of Relation R0.

Attribute name	*Integrity constraint*	*Value type*	*Storage model*	*Distinct values*	*Size in bytes*
A0	Primary key	Numeric	FS	|R0 |	4
A1, A2	Foreign key	Numeric	FS/DS/RS/RIS	|R0 |/6	4
A3		Numeric	FS/RS/RIS	|R0 |/6	4
A4		Numeric	FS/RS/RIS	|R0 |/100	4
A5, A6, A7, A8, A9		Char. string	FS/DS/RS/RIS	|R0 |/10	20 (avg.)

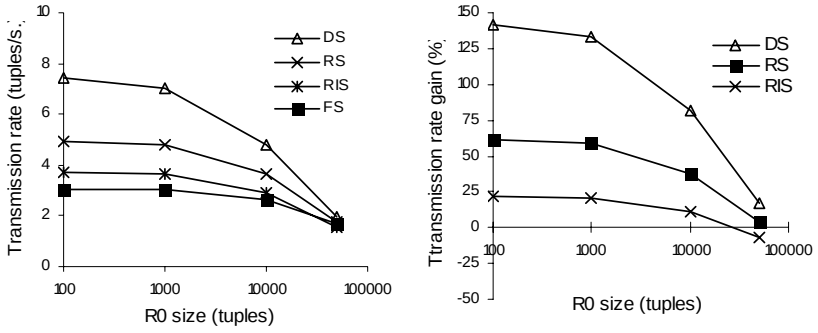
Table 4. Query Set for the Performance Evaluation.

Acronym	*Access right description*	*Corresponding representative query*
P	Subset of attributes	Multi-projection on R0
SP	Predicates on a single relation	Multi-selection on R0
SPJ	Predicates on two relations (direct relationship)	Multi-selection, join between R0 and R1
SPJ^n	Predicates on several relations (transitive relationship)	Multi-selection, joins between R0-R1-R2-R3-R4
SPG	Single relation, mono-attribute aggregation	Mono-attribute group by on R0, one aggregation
SPJG	Several relations, mono-attribute aggregation	Joins between R0, R1, R3, mono-attribute group by, one aggregation
$SPJG^n$	Multi-attribute aggregation	Joins between R0, R1, R3, group by on two attributes of two distinct relations, one aggregation

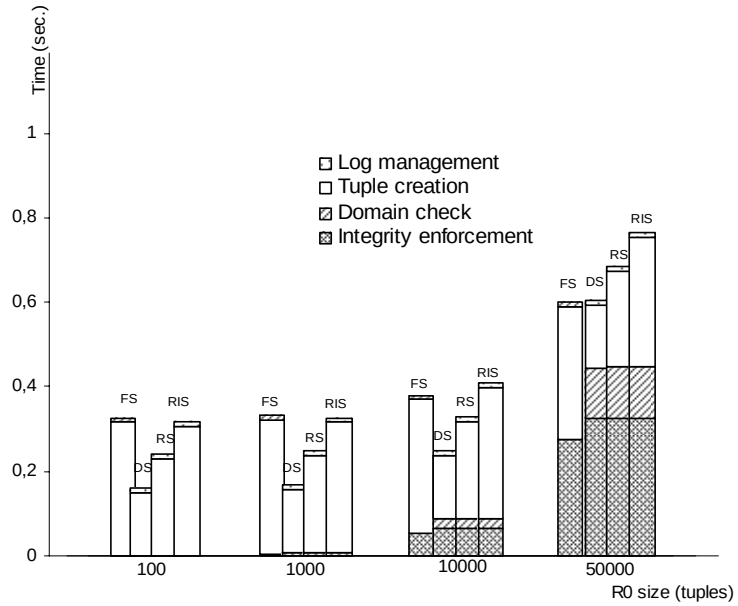
As shown in table 4, the queries measured are representative of the access right classification introduced in Section 2. The attributes on which project, select, join and group by apply are not specified in table 4. The reason for this is that the query pattern is instantiated and measured on different attributes, allowing us to present both the best and worst cases for given query patterns, when significant. The curves presented in the next sections plot performance for different values of project, select and join selectivity, and grouping factor. When not specified, default values are assumed for these parameters: the default projection selectivity is set to 2 attributes (this parameter is actually not a bottleneck), the default selection selectivity is set to 1% of the input relation, default joins are equi-joins on key attributes (i.e., the number of result tuples equals the referencing relation cardinality) and the default grouping factor is set to 1% (i.e., each group aggregates 100 input tuples).

Finally, the remaining of this section will concentrate on the transmission rate metrics. Indeed, while the analogy between the smart card and a magnetic disk (see Section 5.1) is still valid, the latency parameter is not indicated here. Indeed, it can be directly deduced from the transmission rate, the additional treatments exclusively linked to the delivery of the first tuple (building the query execution plan) turning to be negligible in all situations (from 1 to 4 milliseconds depending on the query plan complexity). It strictly corresponds to the time required to build the query plan, the tuples being computed in a pure pipeline fashion.

4.4.3 Tuple Insertion



(a) Insertion rate. (b) Relative performance gain wrt FS.



(c) Decomposed costs.

Figure 12. Performance of Insertions.

Characteristic C2 makes the performance of tuple insertion an important aspect of the evaluation. In the experiment, the tuples are inserted into R0. Most insertions actually occur in R0 since this relation has the biggest cardinality. Moreover, insertion in this relation represents a worst case in terms of performance since it involves the highest integrity control cost: two referential constraints have to be checked and the cost of the uniqueness constraint increases with

the cardinality of the relation. More precisely, a tuple insertion involves the four following steps.

Integrity Enforcement (IE): the target relation (here, R0) must be scanned entirely to check the uniqueness constraint as well as part of each referenced relation to check the referential constraint.

Domain Update (DU): for RS, DS and RIS, the value of each attribute of the inserted tuple has to be searched in the corresponding domain and added if not present. The figures below show the most frequent cost, where the value already exists in the domain (domains turn to be rather stable after a first bunch of insertions).

Tuple Creation (TC): a new chunk of memory is allocated in the target relation and values and pointers are written into it.

Log Management (LM): this operation relates to the pointer-based logging mechanism sketched in Section 3.3.

Figure 12(a) shows that the insertion rate is acceptable (more than one tuple per second) whatever the considered storage model. Figure 12(b) shows that the benefit of DS and RS compared to FS decreases with the cardinality of the relation. While the compactness of the storage model has a positive impact on the write cost in stable storage (write rule), the cardinality of the target relation negatively impacts the IE and DU costs. Note that the IE cost could be reduced thanks to an additional index on the primary key but this would increase in turn the index update cost (writes in stable storage) with a negative effect on small relations and would affect the database footprint. Since transaction throughput is not an issue here, the main conclusion is the one delivered by figure 12(a). Hence, no storage model actually hurt the expected insertion rate, making additional effort to speed-up insertions useless. While Pico-style databases are more insertion than update driven, we did not identify so far insertion intensive scenarios.

4.4.4 Projection

Figure 13 gives the transmission rate for project queries, representative of the P access right type. Curves are plotted for each storage model in function of the number of attributes participating in the projection. The following conclusions can be raised:

The transmission rate is independent of the relation cardinality: obviously, the project operator requires the same time to produce each tuple.

RS slows down the projection: RS induces in average the traversal of half of a ring to reach the value of a given attribute. In our dataset, the average length of rings is 10 pointers, thereby leading to a performance degradation by a factor of 5 (note the logarithmic scale of Figure 13).

The projection is not a bottleneck: the throughput is over 5000 results per second whatever be the considered storage model and the number of projected attributes.

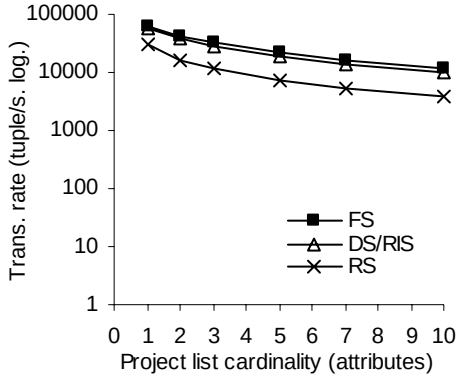
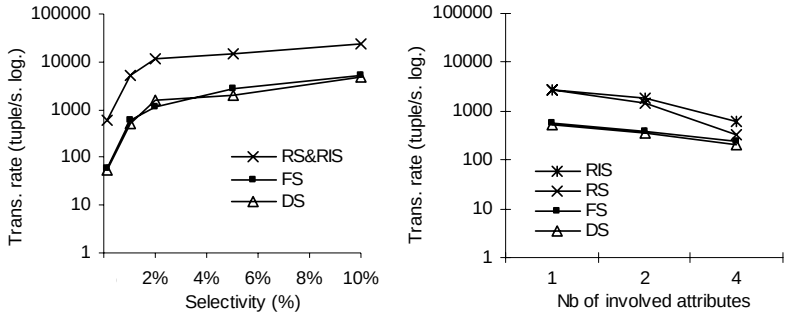


Figure 13. Projection Transmission Rate.

4.4.5 Selection

Figure 14 presents the transmission rate for mono-relation selection queries, representative of the SP access right type. The transmission rate is presented as a function of the query selectivity in figure 14(a) and as a function of the selection complexity (number of attributes involved in the qualification) in figure 14(b). These curves lead to the following remarks.



(a) Mono-attribute Selection. (b) Multi-attribute Selection.

Figure 14. Selection Transmission Rate.

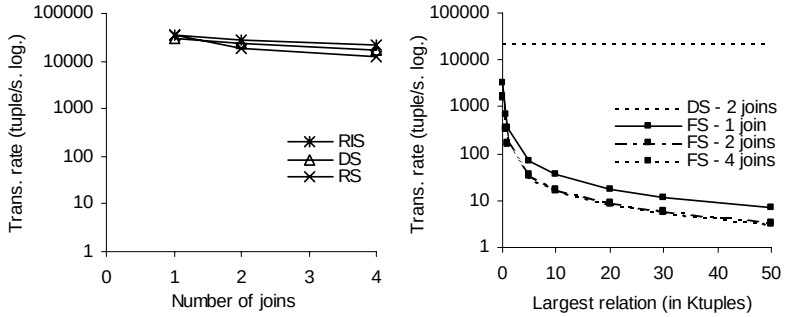
The transmission rate is independent of the relation cardinality: the select operator, similarly to the project operator, requires a constant time to produce each tuple.

The transmission rate benefits from a low query selectivity: indeed, the worse the query selectivity, the less irrelevant tuples scanned and checked before producing a matching tuple.

RIS and RS models outperform DS and FS: RIS and RS offer natural selection indices. The performance gain is directly determined by the ratio between domains and relations cardinalities (set to 10 in our dataset). When the selection involves several attributes, the performance gap between indexed and non-indexed models decreases, highlighting the fact that the PicoDBMS execution model can exploit a single selection index for a given query.

4.4.6 Join

Figure 15 presents the performance of queries representative of SPJ and SPJ^n access right types, involving both selections and joins, with different query selectivity. Figure 15 measures the performance of pure join queries varying the number of joins. These two figures are necessary to capture the incidence of selection on join queries. Let us first consider the conclusions that can be drawn from figure 15.



(a) DS/RS/RIS Model. (b) FS Model.

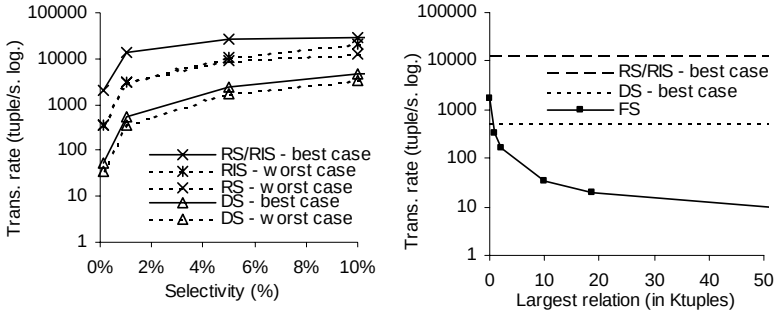
Figure 15. Join Transmission Rate.

High transmission rate for RIS, RS and DS: thanks to their intrinsic indexed nature, these models exhibit a high throughput for pure join queries, regardless of the database size and the

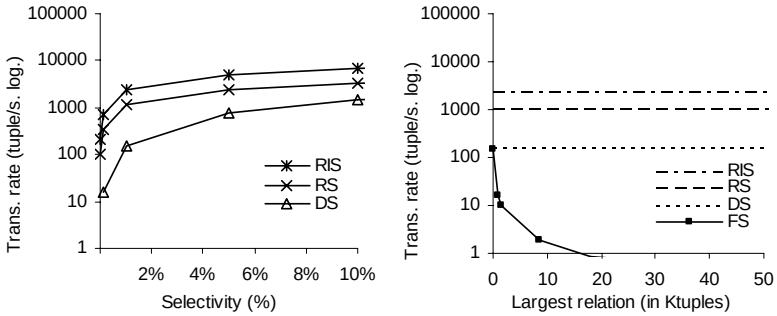
number of joins. As stated in Section 3.1, DS naturally implements a unidirectional join index between a foreign-key attribute and the corresponding primary key attribute. RS and RIS do nothing but making this index bi-directional. The best join ordering privileges traversals from foreign to primary keys, allowing producing each result tuple of a query involving N joins by traversing only N-1 domain pointers. This explains the low performance gap between RIS/RS and DS in the figure. However, this favorable situation cannot always be reached, typically if selections impose selecting another join ordering. This point will be further discussed. Not surprisingly, FS exhibits poor performance comparing to the three other models. Moreover, FS scales very badly as the database size augments (see figure 15(b)).

Let us now consider figure 16, measuring the performance of SPJ and SPJ^n queries. The transmission rate is plotted varying the query selectivity for RIS, RS and DS models (these models have been shown independent of the database cardinality), and varying the database cardinality for FS (with a default selectivity of 1%). For the SPJ^n query pattern, the queries instances may favor different models depending on the relation on which the selections apply. To capture this, figure 16 plots the best and worst cases of the transmission rate for each model.

While self-explanatory, these curves deserve the following remarks. First, the performance of the join increases while the selection selectivity decreases. As for selection queries, the worse the query selectivity, the less irrelevant tuples scanned and checked before producing a matching tuple. Second, RS and RIS clearly outperform DS when selections are considered. As stated before, DS imposes a unique join ordering in the query plan, thereby precluding the use of a selection index. Hence, the choice of RS or RIS to store the foreign key attributes is relevant.



(a) Single Join with Selection (SPJ) – DS/RS/RIS. (b) Single Join with Selection (SPJ) – FS.



(c) Multi-join with Selections (SPJ^n) – DS/RS/RIS. (d) Multi-join with Selections (SPJ^n) – FS.

Figure 16. SPJ and SPJn Transmission Rate.

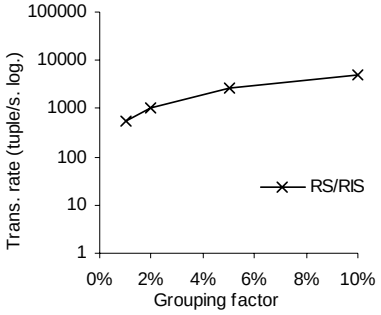
4.4.7 Aggregation

Aggregation queries implement CA authorizations, granting access to computed values without granting access to the occurrences taking part in the computation, a rather usual and important class of authorizations. Figure 17 presents the transmission rates of queries representative

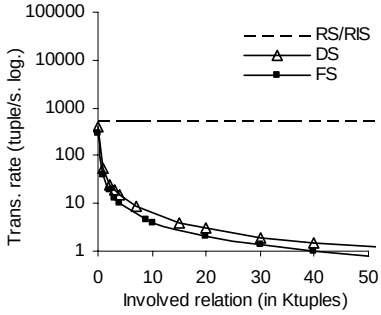
of the SPG, SPJG and SPJG^n access rights (see Table 4). These curves lead to the following remarks:

DS and FS support aggregations badly: the transmission rate is poor for both models, even for mono-attribute aggregations, and strongly depends on the database cardinality. Indeed, DS induces a Cartesian product between the domain the aggregation applies on and the relation, while FS induces successive sequential scans of the relation to produce the result. When joins are combined in the query, the performance of DS remains stable (thanks to a high join delivery) while FS collapses (under one tuple per second).

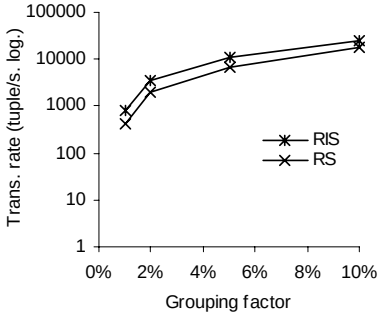
RIS and RS support mono-attribute aggregations gracefully: thanks to rings, the presence of joins in the query has little impact on the performance (see figure 17(a) and figure 17(c)). While the performance is independent of the database cardinality (as for join queries), the grouping factor has a strong influence. For example, a 10% grouping factor (i.e., the aggregation divides the number of tuples by 10), roughly decrease the performance by a factor 10 compared to non-aggregative query.



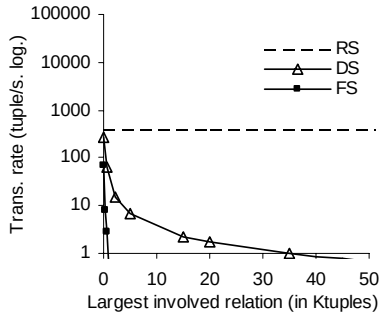
(a) Mono-aggregation (SPG) – RS & RIS.



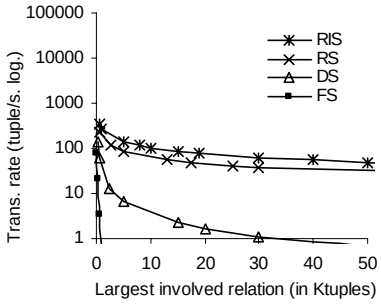
(b) Mono-aggregation (SPG) – FS/DS.



(c) Mono-agg., Joins (SPJG) – RS/RIS.



(d) Mono-agg., Joins (SPJG) – FS & DS.



(e) Multi-agg., Joins (SPJG^n).

Figure 17. Transmission Rate of Aggregation Queries (SPG, SPJG, SPJGn).

RIS and RS support multi-attribute aggregation reasonably: as pictured in figure 17(e), the transmission rate depends now on the database cardinality for all storage models. Indeed, even with RS and RIS, only one of the attributes involved in the group by clause benefits from the ring storage. However, the performance remains acceptable with RS and RIS since more than 80 result tuples per second are still delivered.

4.4.8 Concluding Remarks on Transmission Rates.

To help comparing the FS, DS, RS and RIS models, table 5 expresses their relative performance in terms of ratio for all classes of queries. In each line of table 5, the gray cell serves as a reference to establish the ratios. The transmission rates have been computed considering 1000 tuples in relation R0.

Table 5. Transmission rate ratios.

Operation type \ *Storage model*		*FS*	*DS*	*RS*	*RIS*
Insert query		1	4,8	1,1	0,7
Select query	P	1	0,9	0,5	0,9
	SP	1	0,9	8,7	8,7
	SPJ	0,08	1	23,4	23,4
	SPJ^n	0,01	1	7,1	15,2
	SPG	0,7	1	136	136
	SPJG	0,002	1	118	228
	$SPJG^n$	0,002	1	19	31

DS gives the best transmission rate in terms of insertion and outperforms FS to process queries involving joins (from one to three orders of magnitude). Adding rings provides a major benefit for join and aggregate computations (again an improvement from one to two orders of magnitude over DS). However, the benefit provided by RIS compared to RS is rather disappointing (at most a factor 2 when several joins participate in a query), especially when considering the loss it incurs in terms of database compactness. Nevertheless, note that for particular database schemas, like star schemas where a central large relation is referenced by several small relations through large rings (e.g., hundred of pointers), RIS model could become valuable.

5 Data Management on Chip: Research Directions

PicoDBMS has been designed to provide an effective solution to secured portable folders on smart cards. This section shows that database components (not limited to PicoDBMS) embedded on secured chips (not limited to smart cards) can be exploited in other important contexts and can open very exciting research perspectives.

Section 5.1 focuses on hardware characteristics of secured chips and on their impact on the design of embedded DBMS components (storage, indexation, query execution, etc.). Section 5.2 studies how to extend the sphere of confidentiality of secured chips towards external (unsecured) resources. Section 5.3 considers the huge problem of confidentiality raised by the invasion of smart objects in our everyday life. Finally, Section 5.4 addresses some issues raised by the management of embedded XML data.

5.1 Research Perspectives Driven by Hardware Constraints

Section 2.2 gave an insight of the diversity of secured chips. Whatever their form factor, environment and objectives, secured chips share similar hardware constraints. Indeed, the chip size needs to be reduced for (i) security reasons (to render the cost of physical attacks prohibitive), (ii) economical considerations (to reduce the production cost, hence the silicium die size, on mass-market), and (iii) environmental constraints (to ease the chip integration in domestic ap-

pliances, in smart objects and even in the human body in medical applications). All the hardware resources (CPU, ROM, RAM, stable storage, etc) compete on the same silicium die and they can be balanced in different ways to cope with different requirements. Hence, co-design strategies are required to calibrate a hardware platform according to given application requirements. In the following, we focus on some important hardware constraints, and mention interesting co-design issues.

5.1.1. RAM Resource

RAM is a crucial resource in secured chips because of its poor density. Chip manufacturers usually privilege stable storage to the detriment of a RAM strictly calibrated to hold the execution stack required by on-board programs. Unfortunately, RAM is also a crucial resource wrt the evaluation of database queries. PicoDBMS has been designed to cope with this issue, according to the RAM rule. The solution proposed relies on an intensive use of indices. While this solution is convenient for the targeted context, indices may be disqualified in other contexts because they increase update cost, they make update atomicity more complex to implement and they competes with on-board data. This may cause problems for update intensive applications (e.g., sensed data). In addition, throwing away indices may help reducing the database footprint and the code complexity (thereby the hardware resources in a co-design perspective) in contexts where performance is not the major issue.

In [16] we studied how to minimize the RAM consumption in the query execution, assuming no indices. The objective was precisely to follow a co-design approach wrt the RAM amount and the required performance. First, we designed a RAM lower bound query execution model, concentrating on the algorithmic structure of each relational operator and on the way the dataflow between these operators must be organized. This model induces several iterations on the data to compute the query result. Second, we devised a new form of optimization, called iteration filter that drastically reduces the prohibitive cost incurred by the preceding model, without hurting this RAM lower bound. Finally, we analyzed how to benefit from an incremental growth of the RAM amount. This analysis provides valuable information to determine: how much RAM should be added to reach a given response time, how much the expected response time should be relaxed to tackle a given query with a given quantity of RAM, how much data can be embedded in a given device without hurting an expected execution time.

This work can be considered as a first step towards more complete co-design works aiming at calibrating all resources of a hardware platform.

5.1.2. Stable Memory

In the PicoDBMS study, we focus on EEPROM technology because current smart cards are endowed with this technology. However, many alternative stable memories could be envisioned for secured chips. We sketch here interesting research issues raised by the memory technologies mentioned in Section 2.1. (i.e., FLASH memory, mix of FLASH and EEPROM, long term alternatives).

FLASH memory could become an effective short term alternative for secured chips due to its compactness. FLASH memory can be read and write at a fine grain (e.g., bytes or small pages), but erasure can only be done at a coarse grain (e.g., large blocks). In addition, there is more than one order of magnitude between erasure and write operation and another one between write and read. These specific properties should be taken into account when designing embedded database components and specifically when designing the data storage and indexation models. Indeed, the reduced RAM of secured chips advocates for a highly indexed storage model while the high cost of coarse grain erasure in FLASH advocates for a sequential storage model.

Secured chips could also be endowed with a combination of FLASH and EEPROM. In this context, the performance of database operations is drastically impacted by the storage model and the placement of the indices, data and metadata in each memory. A co-design study should

be particularly helpful to determine (i) the balance between both types of memory and (ii) the FLASH characteristics (e.g., *page* and *block* size).

Long term memory alternatives could also be envisioned, including technologies like PCM, OUM and Millipedes. These memories also provide very specific and interesting access properties which directly impact the storage and indexation model. There are already some works on Millipedes [71] in a classical database context. Combining the memory properties with the other hardware constraints will undoubtedly generate interesting problems.

5.1.3. Cache Conscious Strategies

Another interesting research issue is to improve the processor data cache usage when processing queries. Indeed, some announced smart cards are now endowed with a cache holding up to 1 KB. Since query execution with a constrained RAM induces numerous iterations on the data (e.g., to perform group by, joins without index), cache conscious algorithms could bring great performance improvements, diminishing the overheads of these iterations.

Cache conscious strategies have been envisioned on traditional and main-memory DBMS and led to reorganize indices storage [72] and data layout [73][74]. Recently, cryptographic algorithms requiring high throughput have been implemented in a cache conscious manner [76]. The challenge is that cache conscious processing generally leads to rethink data and index storage organization in a way which may contradict other concerns like data compactness and specific storage models dedicated to new stable memories.

Co-design could again be helpful to help calibrating the processor cache and the RAM, both competing on the same silicium die.

5.1.4. Contactless Interactions

Contactless interfaces are more and more promoted by constructors and governmental organizations for their ease of use [26]. While the current PicoDBMS design could adapt contactless smart cards, whether the processing requirements (in terms of query types and execution time) remain the same is an important question. For instance, severe response time constraints have to be enforced to avoid the card holder to stop when passing near a contactless reader. A possible solution could be to reduce the completeness/accuracy of the result for applications accepting partial results (e.g., top queries, approximate answers).

5.2. Extending the Sphere of Confidentiality to External Resources

Secured chips are often plugged into or linked to more powerful devices (*e.g.*, cell phone, PDA, PC, and even servers). Thus, it does make sense to take advantage of the device resources to overcome the secured chip limitations. The idea is to extend the sphere of confidentiality provided by the security properties of the chip to external resources. This is actually the approach followed by the TCPA architecture where the secured chip protects the whole PC platform. This section sketches how to extend the sphere of confidentiality to external (unsecured) storage resources (section 5.2.1), to external processing resources (section 5.2.2) and finally to a shared database server (section 5.2.3).

5.1.1 External Storage Resources

One of the main limiting factor of traditional smart cards to address new domains of applications is the tiny capacity of the stable memory. To tackle this important issue, several smart card manufacturers are pushing new architectures combining the security of smart cards with very large (but insecure) FLASH memory modules. For instance, the Gemplus' SUMO "smart card" [75] provides 224 MB of FLASH memory dispatched within the plastic substrate; the X-Mobile Card [77] combines in a MMC form factor a secure smart card chip and a large FLASH

memory. These efforts are supported by the MObile PASSport (MOPASS) consortium[13]. Up to 8GB of stable storage are expected in the coming years for these architectures.

In this context, the problem is to use this external storage without losing the traditional security of smart cards. The external storage being insecure, four categories of attacks must be considered: (1) *Data Snooping* (read or infer forbidden data); (2) *Data Altering* (modify - even randomly - forbidden data); (3) *Data Substituting* (replace valid data with another valid data); and (4) *Data Replaying* (replace valid data with an older version of the same data). The device owner himself can attempt to tamper his own data (e.g., to replay a medical act refund using his medical folder). To prevent from information disclosure and protect the data integrity, cryptographic techniques (e.g., encryption, secure hash functions, etc.) must be employed. Some metadata (encryption keys, checksums, freshness information, bootstrap, etc.) and the query evaluator itself must remain embedded in the secured chip to deliver only the authorized data to the user. The problem is to combine cryptographic techniques and query execution techniques in a way satisfying three conflicting objectives: efficiency, high security and compliance with the chip hardware resources. In [78], we conduct an introductive study of this problem considering smart cards enhanced with external FLASH memory.

5.1.2 External Processing Resources

Secured chips are often connected to active devices endowed with processing resources largely exceeding those of the chip itself (e.g., a smart card or dongle plugged into a cellular phone, a PDA or a PC). Exploiting these external computing resources can be helpful to manage a huge amount of data, in situations like the one depicted in the preceding section. In addition, this could allow minimizing the chip hardware resources in a co-design perspective. However, externalized processing must be opaque (it should not reveal any information on the data being processed) and tamper resistant (the honesty of the result must be provable).

While cryptographic techniques can provide opacity and tamper resistance for the externalized data, providing these same properties for externalized processing is today an open problem. Indeed, checking the tamper resistance of an externalized processing might be as expensive as the processing itself. Some techniques exist for simple selections (with plaintext data) [79] but the problem remains unsolved for more complex operations. Enforcing the opacity of externalized processing seems even more difficult to reach since some information about the input and output parameters is always disclosed (even if encrypted) and can be exploited by an attacker. The work conducted on Private Information Retrieval (PIR) illustrates well this problem [80].

Delegating external processing on external (encrypted) data is thus a very challenging research perspective.

5.1.3 External Shared Database

The two preceding sections discuss the extension of the sphere of confidentiality towards external storage and processing resources but do not consider the sharing issue. This does not mean that all forms of sharing are precluded but rather that the sharing is under the control of a single chip, the external resources being considered as direct extensions of this chip (e.g., a PicoDBMS delegating storage and processing to a connected device).

Assuming the database is stored on a remote and unsecured server and is shared among different users leads to another spectrum of applications. For instance, one may consider a shared database hosted by an – untrusted – Database Service Provider. The Database Service Provider Model has been studied for the first time in [82], but sharing was not a concern in this study. The sharing issue has been tackled in [83] thanks to an architecture called Chip-Secured Data Access (C-SDA). C-SDA is a client-based security solution where a smart card acts as an incorruptible mediator between a user and a remote encrypted database. The smart card checks the

[13] The MOPASS consortium is composed of 61 companies including the main smart card and chip manufacturers (e.g., Gemplus, Hitachi, Sharp, NEC, etc.). http://www.mopass.info/english/

user's privileges, participates in the query evaluation and decrypts the final result before delivering it to the user. Since then, this hot topic generated several papers including [84][85]. So far, existing studies focused on the confidentiality of the remote data but disregarded tamper-resistance (including opacity, integrity and freshness). Compared to section 5.2.2, enforcing tamper-resistance properties is more complex in the presence of multiple users, each possessing a personal secured chip (e.g., smart card, dongle). Indeed, sensible information like checksums, or freshness information can no longer be hosted by the secured chips due to potential inconsistencies in case of updates. Solutions relying on a secured co-processor at the server side could be investigated and would raise new research challenges.

5.2 Hippocratic Smart Objects

In the ubiquitous world, people are surrounded by smart, networked and interacting objects, aware of people's presence and needs, and adjusting their actions accordingly [86]. As a side effect, data confidentiality can be compromised in a number of ways and people are reluctant to participate in smart environments (e.g., active badge initiatives attest that many employees do not wear the badge tracking their location).

Secured chips can bring guarantees about the confidentiality of the monitored, stored and transmitted data, in the same spirit as Hippocratic DBMS. Hippocratic databases have been introduced in [43] to ensure that personal data are used in strict compliance with the purpose for which the donor of these data gave his consent. More precisely, Hippocratic databases encompass ten principles: Purpose Specification, Consent, Limited Collection, Limited Use, Limited Disclosure, Limited Retention, Accuracy, Safety, Openness, and Compliance. While enforcing these principles is still an open issue in a general setting, our belief is that secured chips can provide accurate solutions in a smart object context, making these objects inherently Hippocratic.

5.3 Embedding XML Database Components

The study presented in this paper has been conducted in the relational context. As XML becomes a de-facto standard to describe heterogeneous data and exchange them among applications, the need for managing XML data on chip arises. For instance, XML may be the appropriate data model for managing portable personal profiles shared among multiple applications (e.g., in a Virtual Home Environment).

Considering the XML data model raises new concerns. First, while many proposals have been made for storing and indexing XML data, we are not aware of XML storage and indexation models dedicated to chips. The complexity comes from the semi-structured and hierarchical nature of the data. Second, existing XML access control models are rule-based rather than view-based [87][88][84]. This introduces the concern of evaluating a rule-based access control policy in a way compatible with the chip hardware constraints. As a first step in this direction, [89] proposed a streaming XML access control evaluator embedded in a smart card. Digital Right Management models [10][41][11] expressing complex conditions on XML metadata introduce new challenges regarding the enforcement of access control policies on streaming – multimedia – objects.

6 Conclusion

Four years after the publication of the – challenging – PicoDBMS design, the objective of this paper was actually to answer the three important questions raised in the introduction.

The first question was whether the hardware and applications evolution changed the initial problem statement. As shown in section 2.1, while smart card hardware does not escape Moore's law, the resource dissymmetry characterizing smart cards (and then dictating the PicoDBMS design) is rather stable for both technological and economical reasons. At the same time, the introduction of (smart card-like) secured chips in usual computing infrastructures are broadening the scope of PicoDBMS applications towards complex secured portable folders,

advanced DRM model enforcement, Hippocratic smart objects. The first contribution of this paper was to revisit the PicoDBMS problem statement in this light, putting the focus on access control management.

The second question was related to the feasibility of the approach. Clearly, the smart card technology (both hardware and software) at our disposal in 2001 was totally unsuitable for data intensive embedded applications [12]. Three years of joint efforts from our team (PicoDBMS optimization and rewriting in C) and from our industrial partner Axalto (new hardware platform, OS adaptation) were necessary to get a convincing prototype. A benchmark dedicated to Pico-style databases has been designed and used to assess the relative performance of candidate storage and indexation data structures, both on a real smart card platform and on a cycle-accurate simulator. By giving hints to select the appropriate storage and indexation structure for a given application according to the volume of embedded data, the required access rights and the expected response time, this performance study constitutes the second contribution of this paper.

The third question was whether PicoDBMS opens up a broad and long term research. Clearly, relational data management on traditional smart card is now well understood (we hope that the current paper participates to this understanding). However, secured chips are invading our everyday life through a wide variety of form factors: contactless cards, smart dongles, mass storage cards, secured user appliance, secured smart objects etc. The hardware technology is evolving accordingly: huge amount of unsecured stable storage, high communication throughput, future (long-term) stable storage technologies etc. New embedded data management techniques need to be devised to cope with this evolution, like managing very large on-board databases, protecting them against new forms of attacks, taking advantage of the communication throughput to externalize data and/or processing. Co-design is undoubtedly a very important issue in this context. Finally, the secured chip can be integrated in an insecure distributed computing system (e.g., a database system) and be the ultimate trusted party the complete security relies on. Hence, we expect that this paper will contribute to the definition of a fascinating research agenda for database techniques on secured chip.

7 Bibliography

[1] Chen, Z. 2000. Java Card Technology for Smart Cards: Architecture and Programmer's guide. Addison-Wesley. 1

[2] Oracle Corp. 2002. Oracle 9i lite: Release Notes - Release 5.0.1.

[3] Karlsson, J., Lal, A., Leung, C., Pham, T. 2001. IBM DB2 Everyplace: A Small Footprint Relational Database System. In *Proceedings of the International Conference on Data Engineering*.

[4] Seshadri, P., Garrett, P. 2000. SQLServer For Windows CE – A Database Engine for Mobile and Embedded Platforms. In *Proceedings of the International Conference on Data Engineering*.

[5] Sybase Inc. 2000. The Next Generation Database for Embedded Systems. White paper. http://www.ianywhere.com/downloads/whitepapers/embedded.pdf

[6] Pucheral, P., Bouganim, L., Valduriez, P., Bobineau, C. 2001. PicoDBMS: Scaling down Database Techniques for the Smart card. *Very Large Data Bases Journal 10(2-3).*

[7] Mastercard Inc. 2002. MasterCard Open Data Storage (MODS). https://hsm2stl101.mastercard.net/public/login/ebusiness/smart_cards/one_smart_card/biz_opportunity/mods

[8] TCPA. Trusted Computing Platform Alliance. http://www.trustedcomputing.org/

[9] SmartRight. The SmartRight Content Protection System. http://www.smartright.org/

[10] XrML. XrML eXtendible rights Markup Language. http://www.xrml.org/.

[11] ODRL. The Open Digital Rights Language Initiative. http://odrl.net/.

[12] Anciaux, N., Bobineau, C., Bouganim, L., Pucheral, P. 2001. PicoDBMS: Validation and Experience. In *Proceedings of the International Conference on Very Large Data Bases*.

[13] Anciaux, N., Bouganim, L., Pucheral, P. 2003. Database Components on Chip. *ERCIM news 54*.
[14] CR80 News. 2004. New report points way for profitable smart card IC manufacturing. Newsletter. http://www.secureidnews.com/news/2004/07/06/new-report-points-way-for-profitable-smartcard-ic-manufacturing/
[15] Potonniée, O. 2004. A decentralized privacy-enabling TV personalization framework. In *Proceedings of the European Conference on Interactive Television*.
[16] Anciaux, N., Bouganim, L., Pucheral, P. 2003. Memory Requirements for Query Execution in Highly Constrained Devices. In *Proceedings of the International Conference on Very Large Data Bases*.
[17] NRC. 2001. Embedded Everywhere. A Research Agenda for Networked Systems of Embedded Computers. National Academy Press.
[18] Gupta, R., Dey, S., Marwedel, P. 1998. Embedded System Design and Validation: Building Systems from IC cores to Chips. In *Proceedings of the International Conference on VLSI Design*.
[19] Schneier, B., Shostack, A. 1999. Breaking up is hard to do: Modeling Security Threats for Smart Cards. In *Proceedings of the USENIX Symposium on Smart Cards*.
[20] ST Microelectronics. 2004. ST's Trusted Platform Module Provides Complete TCG-Enabled Security Solution for Desktop and Laptop PCs. http://www.st.com/stonline/press/news/year2004
/p1499m.htm
[21] MIPS Technologies Inc. 2002. Smart Cards: the computer in your wallet. White Paper. http://www.mips.com/content/PressRoom/TechLibrary/techlibrary
[22] Maloney, D. L. 2001. Card Technology in Healthcare. In *Proceedings of the CardTech/SecurTech*.
[23] Netc@rd project. http://www.netcards-project.com/new/
[24] NIST. 2002. U. S. Government Smart Card Interoperability Specification GSC-IS, v2.0. Internal Report 6887.
[25] Moriyama, S., Tanabe, H., Sasaki, S., Toyomura, S. 2004. Traceability and Identification Solutions for Secure and Comfortable Society. Hitachi White Paper. http://www.hitachi.com/ICSFiles/afieldfile/2004/05/24/r2004_mar_001.pdf
[26] SINCE. Secure and Interoperable Networking for Contactless in Europe. 2002. Interoperable European Electronic ID / Public Service Cards. Project Deliverable.
[27] NETLINK. 2000. http://www1.va.gov/card/docs/netlink_cook-book_21.pdf
[28] US-GAO. United States General Accounting Office. 2003. Electronic Government: Progress in Promoting Adoption of Smart Card Technology. http://www.gao.gov/new.items/d03144.pdf
[29] CardTechnology. 2003. Omaha Hospitals To Accept Patient Smart Card. Press Release. http://www.cardtechnology.com/cgi-bin/readstory.pl?story=20031104CTDN157.xml
[30] Kim, W. 2004. Smart Cards: Status, Issues, US Adoption. *Journal of Object Technology 3(5)*.
[31] Veterans affairs. 2001. G-8 Healthcare Data Card Project. http://www.va.gov/card/
[32] NETLINK. 2000. http://www1.va.gov/card/docs/netlink_requirements_for_interoperabilityv21.pdf
[33] APEC. Telecommunications and Information Working Group. 2003. Policy and Regulatory Update of Hong Kong, China. http://unpan1.un.org/intradoc/groups/public/documents/APCITY/UNPAN008982.pdf
[34] Henderson, N. J., White, N. M., Hartel, P. H. 2001. iButton Enrolment and Verification Requirments for the Pressure Sequence Smart Card Biometric. In *Proceedings of the International Conference on Research in Smart Cards*.
[35] Vogt, H., Rohs, M., Kilian-Kehr, R. 2003. Middleware for Communications, Chapter 16: Middleware for Smart Cards. John Wiley and Sons.

[36] OCR HIPAA Privacy. 2003. General Overview of Standards for Privacy of Individually Identifiable Health Information.
[37] Smart Card Alliance. 2003. HIPAA Compliance and Smart Cards: Solutions to Privacy and Security Requirements. http://www.smartcardalliance.org/
[38] Smart Card Alliance. 2003. Privacy and Secure Identification Systems: The Role of Smart Cards as a Privacy-Enabling Technology. http://www.smartcardalliance.org/
[39] IBM Harris. 2000. Multinational Consumer Privacy Study. http://www.pco.org.hk/english/infocentre/files/westin.doc
[40] IFPI. International Federation of Phonographic Industry. http://www.ifpi.org/
[41] MPEG-REL. 2004. MPEG-21 Right Expression Language (MPEG-REL), ISO/IEC 21000-5:2004 standard. http://www.contentguard.com/MPEGREL_home.asp
[42] Bouganim, L., Dieu, N., Pucheral, P. 2005. MobiDiQ: Mobile Digital Quietude. Gold Award of the *Simagine 2005 International Contest* (more than 300 participating teams), organized by Sun, Axalto and Samsung.
[43] Agrawal, R., Kiernan, J., Srikant, R., Xu, Y. 2002. Hippocratic Databases. In *Proceedings of the International Conference on Very Large Databases*.
[44] Singhal, V., Kakkad, S., Wilson, P. 1992. Texas: An Efficient, Portable Persistent Store. In *Proceedings of the International Workshop on Persistent Object Systems*.
[45] Heytens, M., Listgarten, S., Neimat, M., Wilkinson, K. 1994. Smallbase: A Main-Memory DBMS for High-Performance Applications. HP Lab. Technical Report.
[46] Chaudhuri, S., Weikum, G. 2000. Rethinking Database System Architecture: Towards a Self-Tuning RISC-Style Database System. In *Proceedings of the International Conference on Very Large Data Bases*.
[47] Olson, M. A. 2000. Selecting and Implementing an Embedded Database System. *IEEE Computer Magazine 33(9).*
[48] Ortiz, J. R. S. 2000. Embedded Databases Come out of Hiding. *IEEE Computer Magazine 33(3).*
[49] Pervasive Software Inc. 2002. Pervasive.SQL v8. http://www.pervasive.com.
[50] Empress Software Inc. 2003. Empress Product profile. http://www.empress.com
[51] Olson, M. A., Bostic, K., Seltzer, M. I. 1999. Berkeley DB. In *Proceedings of the USENIX Conference.*
[52] IBM Corp. 1999. DB2 Everywhere – Administration and Application Programming Guide. IBM Software Documentation SC26-9675-00.
[53] Intanagonwiwat, C., Govindan, R., Estrin, D. 2000. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings of the International Conference on Mobile Computing and Networking*.
[54] Madden, S., Hellerstein, J. M., Hong W. 2004. TinyDB: In-Network Query Processing in TinyOS. *Tutorial at International Conference on Data Engineering*.
[55] Yao, Y., Gehrke, J. 2002. The Cougar Approach to In-Network Query Processing in Sensor Networks. *SIGMOD Record 31(3).*
[56] Bonnet, P., Seshadri, P. 2000. Device Database Systems. In *Proceedings of the International Conference on Data Engineering*.
[57] Bonnet, P., Gehrke, J., Seshadri, P. 2001. Towards Sensor Database Systems. In *Proceedings of the International Conference on Mobile Data Management*.
[58] Madden, S., Franklin, M.J., Hellerstein, J., Hong, W. 2002. TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks. In *Proceedings of the International Conference on Operating Systems Design and Implementation*.
[59] Madden, S., Hellerstein, J. M. 2002. Distributing Queries over Low-Power Wireless Sensor Networks. In *Proceedings of the International Conference on Management of Data*.
[60] International Standardisation Organization. 1999. Integrated Circuit(s) Cards with Contacts - Part 7. ISO/IEC 7816-7.

[61] Bolchini, C., Salice, F., Schreiber, F., Tanca, L. 2003. Logical and Physical Design Issues for Smart Card Databases. *ACM Transactions on Information Systems 21(3)*.
[62] Vingralek, R. 2002. Gnatdb: A small-footprint, secure database system. In *Proceedings of the International Conference on Very Large Databases*.
[63] Missikoff, M., Scholl, M. 1983. Relational Queries in a Domain Based DBMS. In *Proceedings of the International Conference on Management of Data*.
[64] Ammann, A. C., Hanrahan, M., Krishnamruthy, R. 1985. Design of a memory resident DBMS. In *Proceedings of the IEEE International Conference Compcon*.
[65] Pucheral, P., Thevenin, J.-M., Valduriez, P. 1990. Efficient main memory data management using the DBGraph storage model. In *Proceedings of the International Conference on Very Large Data Bases*.
[66] Valduriez, P. 1987. Join Indices. *ACM Transactions on Database Systems 12(2)*.
[67] Shekita, E., Young, H., Tan, K. L. 1993. Multi-Join Optimization for Symmetric Multiprocessors. In *Proceedings of the International Conference on Very Large Data Bases*.
[68] Graefe, G. 1993. Query Evaluation Techniques for Large Databases. *ACM Computing Surveys 25(2)*.
[69] Abdallah, M., Guerraoui, R., Pucheral, P. 2002. Dictatorial Transaction Processing: Atomic Commitment Without Veto Right. In *Proceedings of the International Conference on Distributed and Parallel Databases*.
[70] TPC. Transaction Processing Database Council. http//:www.tpc.org
[71] Yu, H., Agrawal, D., El Abbadi, A. 2003. Tabular Placement of Relational Data on MEMS-based Storage Devices. In *Proceedings of the International Conference on Very Large Data Bases*.
[72] Rao, J., Ross, K.A. 1999. Cache Conscious Indexing for Decision-Support in Main Memory. In *Proceedings of the International Conference on Very Large Data Bases*.
[73] Ailamaki, A. G., DeWitt, D. J., Hill, M. D. 2002. Data Page Layouts for Relational Databases on Deep Memory Hierarchies. In *Proceedings of the International Conference on Very Large Data Bases*.
[74] Ailamaki, A. G., DeWitt, D. J., Hill, M. D., Skounakis, M. 2001. Weaving Relations for Cache Performance. In *Proceedings of the International Conference on Very Large Data Bases*.
[75] Gemplus. 2002. A 224MB microprocessor Smart Card. http://www.gemplus.com/smart/enews/st3/sumo.html
[76] Atasu, K., Breveglieri, L., Macchetti, M. 2004. Efficient AES implementations for ARM based platforms. In *Proceedings of the ACM Symposium on Applied Computing*.
[77] Renesas Tec. 2004. X-Mobile Card Overview. http://www.x-mobilecard.com/products/technology.jsp
[78] Anciaux, N. 2004. Database Systems on Chips. PhD Thesis, Université de Versailles, France.
[79] Gertz, M., Kwong, A., Martel, C., Nuckolls, G., Devanbu, P., Stubblebine, S. 2004. Databases that tell the Truth: Authentic Data Publication. *Bulletin of the Technical Committee on Data Engineering 7(1)*.
[80] Chor, B., Goldreich, O. , Kushilevitz, E., Sudan, M. 1995. Private information retrieval. In *Proceedings of the Annual Symposium on Foundations of Computer Science*.
[81] Damiani, E., De Capitani Vimercati, S., Jajodia, S., Paraboschi, S., Samarati, P. 2003. Balancing Confidentiality and Efficiency in Untrusted Relational DBMSs. In *Proceedings of the ACM Conference on Computer and Communications Security*.
[82] Hacigumus, H., Iyer, B., Li, C., Mehrotra, S. 2002. Executing SQL over Encrypted Data in the DSP Model. In *Proceedings of the International Conference on Management of Data*.
[83] Bouganim, L., Pucheral, P. 2002. Chip-Secured Data Access: Confidential Data on Untrusted Servers. In *Proceedings of the International Conference on Very Large Data Bases*.
[84] Damiani, E., De Capitani di Vimercati, S., Paraboschi, S., Samarati, P. 2002. A Fine-Grained Access Control System for XML Documents. *In Proceedings of the ACM TISSEC*.

[85] Iyer, B., Mehrotra, S., Mykletun, E., Tsudik, G., Wu, Y. 2004. A Framework for Efficient Storage Security in RDBMS. In *Proceedings of the International Conference on Extending Database Technology*.
[86] Weiser M. 1991. The Computer for the Twenty-First century. *Scientific American*.
[87] Bertino, E., Castano, S., Ferrari, E. 2001. Securing XML documents with Author-X. *IEEE Internet Computing 5(3)*.
[88] Gabillon, A., Bruno, E. 2001. Regulating access to XML documents. In *Proceedings of the IFIP Working Conference on Database and Application Security*.
[89] Bouganim, L., Dang Ngoc, F., Pucheral, P. 2004. Client-Based Access Control Management for XML documents. In *Proceedings of the International Conference on Very Large Data Bases*.
[90] Bobineau, C., Bouganim, L., Pucheral, P., Valduriez, P. 2000. PicoDBMS: Scaling down Database Techniques for the Smart card. In *Proceedings of the International Conference on Very Large Data Bases*. (Best Paper Award).